

ARBEITSBERICHTE DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG

FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG

**HERAUSGEBER: H. Billing, W. Händler, U. Herzog,
F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees,
H. Niemann, D. Seitzer, H. J. Schnelder**

Zweistufige Graphmanipulationssysteme für die Semantik von Programmiersprachen

Herbert Göttler

Dissertation

BAND 10 • NUMMER 12 • ERLANGEN • JULI 1977

1977 # 269

ARBEITSBEREICHTE
DES INSTITUTS FÜR MATHEMATISCHE
MASCHINEN UND DATENVERARBEITUNG
BREMSEN- UND BREMSZUG-RECHNUNG
VON DR. H. K. 1954
1. HEFT, 1. LAF, 1. HEFT, 1. LAF, 1. HEFT, 1. LAF
1. HEFT, 1. LAF, 1. HEFT, 1. LAF, 1. HEFT, 1. LAF



Digitized by the Internet Archive
in 2026

ARBEITSBERICHTE

DES INSTITUTS FÜR MATHEMATISCHE MASCHINEN UND DATENVERARBEITUNG

FRIEDRICH ALEXANDER UNIVERSITÄT ERLANGEN NÜRNBERG

**HERAUSGEBER : H. Billing, W. Händler, U. Herzog,
F. Hofmann, K. Leeb, P. Mertens, H. Müller, G. Nees,
H. Niemann, D. Seitzer, H.J. Schnelder**

Zweistufige Graphmanipulationssysteme für die Semantik von Programmiersprachen

Herbert Göttler

Dissertation

BAND 10 • NUMMER 12 • ERLANGEN • JULI 1977

ARBEITSBERICHT
DES INSTITUTS FÜR MATHEMATISCHE
MASCHINEN UND DATENVERARBEITUNG
FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
VERGLEICHENDE MATHEMATIK: W. HILDEBRAND, U. HENNIG,
F. HENNIG, K. KOSCH, R. MATHIAS, H. REISS, G. WIEBE,
H. WILHELM, D. ZIEGLER, H. J. SCHNEIDER

Zweistufige Graphmanipulationssysteme für
die Semantik von Programmiersprachen



Herbert Götter

Als Dissertation genehmigt vom
Fachbereich Ingenieurwissenschaften
der Universität Erlangen-Nürnberg

Tag der Einreichung:	4. Mai 1977
Tag der Promotion:	6. Juli 1977
Dekan:	Prof. Dr. H. J. Schneider
Berichterstatter:	Prof. Dr. H. J. Schneider Prof. Dr. H. Müller

Dissertation

BAND 10 - NUMMER 12 - ERGÄNZUNG JULI 1977

ZUSAMMENFASSUNG :

Graphische Darstellungen haben sich zur Präsentation eines Problems und seiner Lösung als nützlich erwiesen. Ihr Nachteil: Es fehlen Mittel für ihre formale Handhabung. In diesem Papier wird dargestellt, wie sich Graphen für die Beschreibung der Semantik von Programmiersprachen einsetzen lassen. Dies wird anhand eines Beispiels gezeigt, indem jeder Regel einer Zweistufengrammatik, die die Syntax beschreibt, eine Graphmanipulationsregel zugeordnet wird, die die Semantik der Regel beschreibt. Da dies sowohl für die Hyper- als auch für die Metaebene geschieht, ergibt sich die Notwendigkeit, Zweistufengraphmanipulationssysteme einzuführen, die das Problem angehen, wie man Graphmanipulationsregeln konsistent manipuliert, die nach ihrer eigenen Manipulation die Manipulation von Graphen erlauben. Die Idee ist, Graphmanipulationsregeln selbst als Graphen zu betrachten, um nur einen einzigen Mechanismus für die Manipulation der betrachteten Graphen und der Graphmanipulationsregeln selbst einführen zu müssen. Die Reihenfolge der Manipulationen wird durch das Ergebnis der syntaktischen Analyse des Programmtextes bestimmt.

ABSTRACT :

Graphical representations have proved to be very useful for the presentation of a problem and for its solution. Their disadvantage: The lack of means for handling them formally. This paper describes how graphs can be used for the description of the semantics of a programming language. An example is given, in which to each rule of a two-level grammar, describing the syntax, a graph-manipulation-rule is associated, describing the semantics. Since this has to be done for the hyper- and meta-level as well, the necessity arises to introduce two-level graph-manipulation-system, which cope with the problem how to manipulate graph-manipulation-rules consistently, which, after their own manipulation, allow to manipulate graphs. The idea is to look upon graph-manipulation-rules as being graphs themselves, in order to have to introduce just one mechanism for the manipulation of the considered graph and for the manipulation-rules themselves. The order of manipulation is controlled by the result of the syntactic analysis of the program text.

INHALTSVERZEICHNIS

Seite

1	Motivation und Einführung in die Problematik	1
2	Graphentheoretische Hilfsmittel	6
	2.1 Graphentheoretische Grundbegriffe	7
	2.2 Hierarchische Graphen	15
3	Quasihierarchische Graphen	20
4	Graphmanipulationsvorschriften	25
5	Einstufige und zweistufige Graphmanipulations- systeme	51
6	Manipulationssysteme für quasihierarchische Graphen	77
7	Probleme der semantischen Beschreibung	92
8	Semantikbeschreibung durch Manipulations- systeme für quasihierarchische Graphen; Beispiele	96
9	Weitere Probleme	125
10	Vergleich mit anderen Arbeiten	129
	10.1 Andere Graphmanipulationsansätze	130
	10.2 Andere Semantikbeschreibungsansätze	136
11	Schlußbemerkung	139
12	Stichwortverzeichnis	140
13	Literaturverzeichnis	144

1 MOTIVATION UND EINFÜHRUNG IN DIE PROBLEMATIK

Am Anfang dieser Arbeit stand das immer wieder beobachtbare Phänomen, daß man mitunter trotz hohen verbalen Einsatzes nicht in der Lage ist, seinen Zuhörern einen neuen Sachverhalt näherzubringen. Gibt man jedoch eine graphische Darstellung dazu an, wird häufig das Problem verstanden.

Nun soll keine Deutung des Phänomens durch Spekulationen über etwaige menschliche Gehirnstrukturen versucht, sondern nur ein Beispiel dafür angegeben werden, das geradezu paradigmatisch in diesem Zusammenhang ist: ALGOL 68. In [73] (und später in [74] präziser gefaßt) wird diese Programmiersprache definiert. Jener "Report" gilt bei manchen Fachleuten praktisch als unverständlich und nur für Esoteriker bestimmt.

Dieser Auffassung konnten sich auch die Autoren von ALGOL 68 nicht ganz entziehen und es folgte quasi als Begleitheft die "Informal Introduction" ([34]). Man kann vieles an diesem Buch aussetzen, aber den Verfassern gelingt es, hauptsächlich durch den Einsatz von graphischen Darstellungen, selbst schwierige Sachverhalte, d. h. die Semantik spezieller Sprachkonstruktionen, verhältnismäßig einfach zu erläutern.

In [53] wird von RAIN Kritik geübt. Er vertritt die Meinung, daß die momentanen Methoden zur formalen Definition von Programmiersprachen unbrauchbar sind - symptomatisch dafür sind für ihn das Erscheinen der Wiener Definitionssprache (VDL) [72] und die bereits erwähnte Notwendigkeit, dem "Report" die "Informal Introduction" anzuschließen -. Was RAIN beim "Report" zu formal ist, wird ihm im letzteren zu oberflächlich behandelt: "... lacks any semblance of the rigor desired in a definition". Man muß zustimmen, denn bei der Darstellung der ALGOL 68-Semantik durch formalisierte Bilder ist hauptsächlich die Komposition der Bilder informell gehalten. Sie beruht meist auf bloßer Intuition und erfordert viel Vorverständnis für Programmiersachverhalte beim Betrachter.

Das bedeutet, daß ein Leser nicht in der Lage ist, ausschließlich durch die Anwendung von angegebenen Regeln, sich bei der Klärung eines Problems das richtige Bild, das es löst, selbst zu zeichnen.

Entschließt man sich dazu, mithilfe von Graphen und graphischen Darstellungen auch komplexe Probleme zu beschreiben und zu lösen und weist man nicht sofort die Möglichkeit von der Hand, daß man mit zweidimensionalen Zeichenketten, also graphischen Darstellungen, genauso formal korrekt arbeiten kann, wie man es in der Mathematik vom Umgang mit eindimensionalen Zeichenketten^{*)} gewohnt ist, so wird man sich schnell bewußt, wie das eingangs erwähnte Phänomen zu nutzen und gleichzeitig Abhilfe für den RAINschen Kritikpunkt zu schaffen ist: Man gebe ein möglichst einfaches Regelsystem an, mit dessen Hilfe sich jeder Leser die zum Verständnis, also hier für die Semantik eines Sprachkonstruktes, notwendige graphische Darstellung auf eindeutige Weise selbst herleiten kann.

Konstruktionsregeln für die Bildung eindimensionaler Zeichenketten finden sich in Büchern über Formale Sprachen, z. B. in [57]. Während die Operation "Ersetze in der Zeichenkette 'abc' das Zeichen 'b' durch 'de!'", was als Ergebnis 'aded' ergibt, keine Schwierigkeiten aufzugeben scheint, gestaltet sich dieses einfache Zeichenersetzungsproblem im Falle einer analogen Aufgabenstellung bei Graphen weitaus schwieriger. Gibt man an, daß im Graphen der Figur 1/1, nun informal gesprochen, die Ecke 1 durch

^{*)} In der Mathematik hat man seit langer Zeit zweidimensionale Zeichenketten benutzt. Man denke nur an das Integral $\int f(x) dx$, an eine Determinante, eine indizierte Größe oder gar an einen einfachen Bruch. Warum die im folgenden diskutierten Probleme dabei nicht auftraten, wird noch klar.

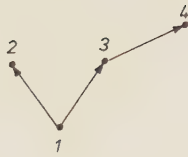


Fig. 1/1

den Graphen der Figur 1/2 ersetzt werden soll, so ergibt

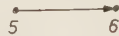


Fig. 1/2

sich das Problem, wie wir die beiden Graphen zu einem neuen "zusammenschließen" sollen (s. a. Figur 1/3).

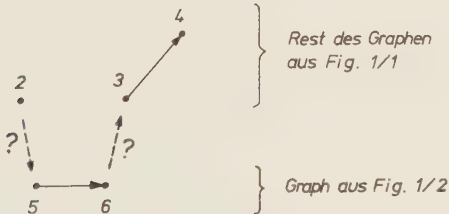


Fig. 1/3

Dieses "An-" oder "Zusammenschließungsproblem", auch "Einbettungs-" oder "Embeddingproblem" genannt, wird im Falle von eindimensionalen Zeichenketten intuitiv durch Auseinanderziehen bzw. Zusammenschieben der Restzeichenketten gelöst; es ist aber das inhärente Problem bei Ersetzungen von Teilstrukturen in Graphen überhaupt. Es besteht also grob gesprochen in der Angabe einer Vorschrift, die auf eindeutige Art und Weise beschreibt, wie der Restgraph, der durch Wegnahme des zu ersetzenden Teiles entsteht, mit dem neu hinzugekommenen Graphen verbunden wird.

Die Literatur auf dem Gebiet der "Graphgrammatiken" ist stark angewachsen, so daß es mehrere Ansätze und Vorschläge zur Lösung des Embeddingproblems gibt. Einer der ersten stammt von SCHNEIDER in [58]. Weitere interessante Darstellungen werden in EHRIG/PFENDER/SCHNEIDER [16] und bei NAGL in [43] beschrieben. In letzterer findet man auch ein ausgedehntes Literaturverzeichnis. Andere momentan existierende Vorschläge zur Embeddingbeschreibung sind im NAGLschen Ansatz fast ausnahmslos als Spezialfälle enthalten.

Auf die o. g. Ansätze werden wir im folgenden noch eingehen, wenngleich die Zielsetzungen und Methoden jener Arbeiten sich von denen der vorliegenden Arbeit unterscheiden. Während es bei den Zielsetzungen der anderen Publikationen hauptsächlich um die bekannten Fragestellungen geht, etwa nach den Eigenschaften der Regeln, die auf einen Startgraphen angewendet eine vorgegebene Klasse von Graphen erzeugen sollen, ist die Zielsetzung der vorliegenden Arbeit mehr anwendungsorientiert: Es wird im folgenden untersucht, wie *Graphen als Beschreibungsmittel* eingesetzt werden können, besonders im Hinblick auf die Semantik von Programmiersprachen. Da sich im folgenden ein modifizierter Begriff von 'Graph' als brauchbar erweist, müssen auch *neue Methoden entwickelt werden, die Umformungen bei Graphen und deren 'Verallgemeinerungen' beschreiben sollen*. Deshalb werden als neue

Objekte 'quasihierarchische Graphen' eingeführt sowie 'einstufige' und 'zweistufige Graphmanipulationssysteme', die eine formale Behandlung dieser Objekte erlauben.

2 GRAPHENTHEORETISCHE HILFSMITTEL

In den beiden folgenden Abschnitten sollen die Grundbegriffe aus der Theorie der gewöhnlichen Graphen eingeführt werden, etwa der Darstellung in [42] folgend, sowie die notwendigen Definitionen zur Einführung von hierarchischen Graphen nach [62] gegeben werden. Da sich die vorliegende Arbeit nicht nur mit Graphen als *Objekten* der Untersuchung, sondern als *Mittel* beschäftigt, fehlen einige Begriffe, die man sonst im Zusammenhang mit "Grundbegriffen der Graphentheorie" erwarten würde.

2.1 GRAPHENTHEORETISCHE GRUNDBEGRIFFE

Sei E_g eine endliche Menge. Ein *bewerteter, gerichteter Graph* g über den Alphabeten $Eckm$ und Kam ist ein Tupel

$g = (\tilde{E}_g, \tilde{K}_g)$, wobei gilt:

$\tilde{E}_g \subseteq E_g \times Eckm$ ist die (markierte) Eckenmenge,

$\tilde{K}_g \subseteq E_g \times E_g \times Kam$ ist die (markierte) Kantenmenge,

E_g ist die Eckenbezeichnungsmenge,

$Eckm$ ist die Eckenmarkierungsmenge und

Kam ist die Kantenmarkierungsmenge.

Die Relation $\tilde{E}_g$ wird als linkstotal und rechtseindeutig vorausgesetzt.

Da wir nur bewertete, gerichtete Graphen betrachten werden, sprechen wir künftig nur von *Graphen*.

Sei g ein Graph über $Eckm$ und Kam , E_g die Eckenbezeichnungsmenge und $e, e' \in E_g$, $a, b \in Eckm$, $m \in Kam$.

Zu g lassen sich leicht die Projektionsfunktionen angeben:

$$\begin{array}{ll} \text{emark}_g : \tilde{E}_g \rightarrow Eckm & \text{mit } \text{emark}_g((e, a)) = a, \\ \text{kmark}_g : \tilde{K}_g \rightarrow Kam & \text{mit } \text{kmark}_g(((e, a), (e', b), m)) = m, \\ \text{ebez}_g : \tilde{E}_g \rightarrow E_g & \text{mit } \text{ebez}_g((e, a)) = e, \\ \text{ae}_g : \tilde{K}_g \rightarrow E_g & \text{mit } \text{ae}_g(((e, a), (e', b), m)) = (e, a), \\ \text{ee}_g : \tilde{K}_g \rightarrow E_g & \text{mit } \text{ee}_g(((e, a), (e', b), m)) = (e', b). \end{array}$$

emark_g ist die Eckenmarkierungsfunktion von g ,

kmark_g ist die Kantenmarkierungsfunktion von g ,

ebez_g ist die Eckenbezeichnungsfunktion von g ,

ae_g ist die Anfangseckenfunktion von g und

ee_g ist die Endeckenfunktion von g .

Wegen der Rechtseindeutigkeit von $\tilde{E}_g$ müssen zur Kennzeichnung einer Kante $\tilde{K} \in \tilde{K}_g$ die Markierungen der Anfangs- bzw. Endecke nicht mit hinzugenommen werden. Wir definieren deshalb:

$$K_m(g) := \{ (e, e', m) \mid (\exists \tilde{k} \in \tilde{K}_g) (e = \tilde{e} \tilde{b} \tilde{e}_g (\tilde{a} \tilde{e}_g (\tilde{k})) \wedge e' = \tilde{e} \tilde{b} \tilde{e}_g (\tilde{e} \tilde{e}_g (\tilde{k})) \wedge \tilde{k} \text{mark}_g (\tilde{k}) = m) \}^* g$$

Somit können wir die Kantenmenge von g auch durch

$$K(g) = \bigcup_{m \in \text{Kam}} K_m(g) \text{ darstellen.}$$

Für $k \in K(g)$ führen wir die Projektionsfunktionen ae_g und ee_g ein:

$$ae_g: K(g) \rightarrow E_g \text{ mit } ae_g((e, e', m)) = e \quad \text{und}$$

$$ee_g: K(g) \rightarrow E_g \text{ mit } ee_g((e, e', m)) = e'.$$

$ae_g(k)$ ist die *Anfangsecke* von k und $ee_g(k)$ die *Endecke*.

Trivialerweise gilt $k = (ae_g(k), ee_g(k), m)$. Zur Vereinfachung lassen wir oft die Indizierungen bei ae_g bzw. ee_g weg und schreiben einfach ae bzw. ee , wenn keine Verwechslung eintreten kann.

Um im Bedarfsfall von der relationalen Schreibweise $\tilde{E}_g$ für die Ecken eines Graphen g abgehen und zu einer funktionalen Schreibweise übergehen zu können, definieren wir eine Funktion $\text{emark}_g: E_g \rightarrow \text{Eck}_m$ durch $\text{emark}_g := \tilde{E}_g$.

Wir werden künftig z. B. statt E_g auch $E(g)$, statt $\tilde{E}_g$ auch $\tilde{E}(g)$ schreiben, um die entsprechenden Mengen zu kennzeichnen.

In den graphischen Darstellungen von Graphen werden die Eckenbezeichnungen häufig nicht oder nicht vollständig angegeben (um erhöhte Übersichtlichkeit zu erreichen), wenn sich keine Unklarheiten ergeben.

Ein Graph kann durch eine Zeichnung veranschaulicht werden, die aus Pfeilen und Kreisen als Grundbausteinen besteht. In den Kreisen stehen die Markierungen der Ecken, außerhalb deren Bezeichnungen. An die die Kanten repräsentierenden Pfeile werden die Kantenmarkierungen hingeschrieben. Trivialerweise enthält nach Definition ein Graph keine Mehrfachpfeile gleicher Richtung und gleicher Markierung.

*) Um Arbeit zu sparen, werden wir auch statt $(e, e', m) \in K_m(g)$ einfach $(e, e') \in K_m(g)$ schreiben.

Beispiel 2.1/1:

Sei $g=(\tilde{E},K)$ ein Graph über $Eckm=\{a,b,c\}$ und $Kam=\{w,x,y,z\}$,
mit $E(g)=\{1,2,3,4,5,6\}$ und $K(g)=K_w(g)\cup K_x(g)\cup K_y(g)\cup K_z(g)$, wobei
 $K_w(g)=\{(4,1),(1,2),(2,3),(1,4)\}$,
 $K_x(g)=\{(2,3)\}$,
 $K_y(g)=\{(1,4),(6,3)\}$,
 $K_z(g)=\{(6,6),(6,4)\}$ ist.

Weiterhin ist

$\tilde{E}(g)=\{(1,a),(2,a),(3,b),(4,c),(5,b),(6,c)\}$ und
 $\tilde{K}(g)=\{((1,a),(2,a),w),((2,a),(3,b),w),((2,a),(3,b),x),\dots\}$

Figur 2.1/1 gibt die graphische Darstellung von g an.

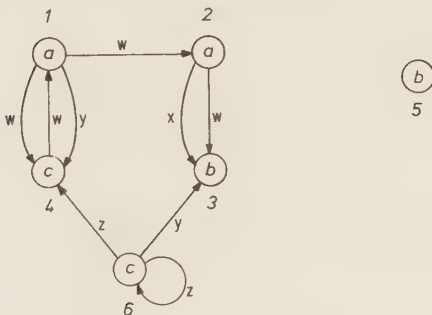


Fig. 2.1/1



Bislang drückte die Schreibweise für Graphen aus, daß wir nur 'zusammenfassende' Ecken- und Kantenmengen zu einem Graphen zusammenfassen sollen. Mithilfe des Operators *con* wollen wir beliebige (endliche) 'Ecken-' und 'Kantenmengen' zu einem Graphen zusammenfassen und dabei eine 'Konsistenz-

prüfung' vornehmen.

Seien B , $Meck$, Mka endliche Mengen, und $M \subseteq B \times Meck$ eine rechtseindeutige Relation, sowie $N \subseteq (B \times Meck) \times (B \times Meck) \times Mka$. Mit $con(M, N)$ bezeichnen wir den aus M und N konstruierbaren Graphen über $Meck$ und Mka , für den gilt:

- 1) $\tilde{E}(con(M, N)) = M$
- 2) $\tilde{K}(con(M, N)) = N \cap (M \times M \times Mka)$.

N läßt auch für unsere Zwecke sinnlose Konstruktionen wie etwa $((1, a), (1, b), n) \in N$ zu. Die Bedingung 2) 'filtert' aber wegen der Rechtseindeutigkeit von M solche 'Kanten' aus und garantiert gleichzeitig, daß nur jene Kanten aus N zum Tragen kommen, für die auch eine Anfangs- und Endecke aus M vorhanden ist. Weiterhin gilt trivialerweise $E(con(M, N)) = B$.

Ist g ein Graph über Eck_m und Kam , dann lassen sich für con folgende triviale Eigenschaften beweisen:

- 1) $con(\tilde{E}(g), \tilde{K}(g)) = g$
- 2) $\tilde{e} \notin \tilde{E}(g) \rightarrow con(\tilde{E}(g) \cup \{\tilde{e}\}, \tilde{K}(g)) = (\tilde{E}(g) \cup \{\tilde{e}\}, \tilde{K}(g))$.
- 3) $\tilde{e} \in \tilde{E}(g) \Rightarrow con(\tilde{E}(g) \setminus \{\tilde{e}\}, \tilde{K}(g)) =$
 $= (\tilde{E}(g) \setminus \{\tilde{e}\}, \{\tilde{k} \mid \tilde{k} \in \tilde{K}(g) \wedge \tilde{e} \neq \tilde{ae}(\tilde{k}) \wedge \tilde{e} \neq \tilde{ee}(\tilde{k})\})$

Sei $m \in Kam$

- 4) $\tilde{e} \notin \tilde{E}(g) \vee \tilde{e}' \notin \tilde{E}(g) \Rightarrow con(\tilde{E}(g), \tilde{K}(g) \cup \{(\tilde{e}, \tilde{e}', m)\}) = g$
- 5) $\tilde{e}, \tilde{e}' \in \tilde{E}(g) \Rightarrow con(\tilde{E}(g), \tilde{K}(g) \setminus \{(e, e', m)\}) = (\tilde{E}(g), \tilde{K}(g) \setminus \{(e, e', m)\})$
- 6) $\tilde{e}, \tilde{e}' \in \tilde{E}(g) \Rightarrow con(\tilde{E}(g), \tilde{K}(g) \cup \{(e, e', m)\}) = (\tilde{E}(g), \tilde{K}(g) \cup \{(\tilde{e}, \tilde{e}', m)\})$.

Die analogen Behauptungen lassen sich leicht auf E , K , ee und ae übertragen, ebenso die Definition für con über die 'verkürzte' Schreibweise für Graphen, bei der wir der Einfachheit halber die Eckenmarkierung bei Angabe einer Ecke nicht explizit angeben (und somit z. B. für 1) auch $con(E(g), K(g)) = g$ genügen würde).

Es 'vergrößert' sich ein Graph durch Hinzunahme einer Ecke und 'verkleinert' sich bei der Wegnahme einer Ecke um diese Ecke und um alle 'daranhängenden' Kanten. Die Hinzunahme einer Kante bleibt unwirksam, wenn die Anfangs- oder die Endecke nicht in $E(g)$ liegen.

Da die Eckenbezeichnungen bei einem Graphen i. a. uninteressant sind und deshalb, wie bereits angedeutet, oft weggelassen werden, geht man mittels einer Äquivalenzrelation über zu einer Klasseneinteilung. In den einzelnen Klassen liegen die Graphen, die die gleiche Gestalt haben und sich lediglich durch die Eckenbezeichnungen unterscheiden. Dazu dienen die folgenden Definitionen, bei denen g und g' Graphen über den Alphabeten Eck_m und Kam sind.

Zuvor noch ein Hinweis: Betrachtet man zwei Abbildungen φ und φ' als Teilmengen des kartesischen Produkts ihres Definitions- und Wertebereichs, so ist die Schreibweise $\varphi \subseteq \varphi'$ sinnvoll.

g' heie *homomorphes Bild* von g (in Zeichen: $g' \underline{hom} g$) g.d.w.

es eine surjektive Funktion $h: E(g) \cup K(g) \rightarrow E(g') \cup K(g')$ gibt mit

1) $h(E(g)) = E(g')$ und

2) $h(K(g)) = K(g')$ und

3) $(\forall e_1, e_2 \in E(g)) (\forall r \in Kam) ((e_1, e_2) \in K_r(g) \rightarrow (h(e_1), h(e_2)) \in K_r(g'))$ und

4) $(e_1', e_2') \in K_r(g') \rightarrow (\exists (e_1, e_2) \in K_r(g)) (h(e_1) = e_1' \wedge h(e_2) = e_2')$ *).

Die Menge der Homomorphismen h von g nach g' bezeichnen wir mit $hom\langle g, g' \rangle$.

Fordert man noch, da obiger (Graph)-Homomorphismus h auch injektiv ist, so nennt man g' ein *isomorphes Bild* von g (in Zeichen: $g' \underline{isog}$).

Gilt $g' \underline{isog}$ und zugleich

$(\forall e \in E(g)) (emark_g(e) = emark_{g'}(h(e)))$,

so heien g' und g *äquivalent* (in Zeichen: $g' \underline{ägg}$).

Gilt $g' \underline{ägg}$, dann wollen wir die Menge der Isomorphismen, die dieser Relation zugrunde liegen, mit $\underline{äq}\langle g', g \rangle$ bezeichnen.

*) Es sind weitere Variationen für die Definition von h möglich, da z. B. Kanten aus g Ecken aus g' zugeordnet werden. Doch auf diese Möglichkeit wird hier verzichtet. Der Einfachheit halber wurde auch gleich verlangt, da Urbild und Bildkanten gleich markiert sind.

Sei $h \in \text{Hom}\langle g, g' \rangle$ ein Graphhomomorphismus und $t = \text{con}(M, N)$ mit $M \subseteq E(g)$ und $N \subseteq K(g)$. Wir bezeichnen die Funktion $\text{restrikt}\langle h, t \rangle \subseteq h$ als die *Restriktion von h auf t* , wenn gilt:

$$\text{restrikt}\langle h, t \rangle = \{ (x, y) \mid x \in E(t) \cup K(t) \wedge y = h(x) \}.$$

Homomorphe Graphen kann man intuitiv als 'ähnlich' bezeichnen, isomorphe Graphen haben bis auf die Eckenbezeichnungen und Eckenmarkierungen die gleiche Gestalt, äquivalente Graphen stimmen bis auf die Eckenbezeichnungen überein. $\underline{\text{äq}}$ ist die gewünschte Äquivalenzrelation. Im folgenden wird meist nur mit Repräsentanten der Äquivalenzklassen gearbeitet und die explizite Eckenbezeichnung nur dann zu einer Beschreibung eines Sachverhaltes hinzugenommen, wenn es unvermeidlich ist.

Der Graph g' heiße *Teilgraph* von g (in Zeichen: $g' \underline{\text{tgg}}$) g.d.w. $\tilde{E}(g') \subseteq \tilde{E}(g)$, $\tilde{K}(g') \subseteq \tilde{K}(g) \cap (\tilde{E}(g') \times \tilde{E}(g'))$. Gilt $g' \underline{\text{tgg}}$ und $\tilde{K}(g') = \tilde{K}(g) \cap (\tilde{E}(g') \times \tilde{E}(g'))$, so nennen wir g' einen *Untergraphen* von g (in Zeichen: $g' \underline{\text{ugg}}$).

Ein Untergraph unterscheidet sich also von einem Teilgraph dadurch, daß, wenn $e_1, \tilde{e}_2 \in \tilde{E}(g')$ und $(e_1, e_2, r) \in \tilde{K}(g)$, dann auch $(e_1, \tilde{e}_2, r) \in \tilde{K}(g')$ ist. Bei der Teilgrapheneigenschaft muß das nicht gelten.

In Analogie zur Benennung der $\underline{\text{äq}}$ zugrunde liegenden Abbildungen wollen wir die (i. a. mehreren) bei der Gültigkeit der Relation $g' \underline{\text{ugg}}$ auf triviale Weise konstruierbaren Funktionen mit $ug \langle g', g \rangle$ bezeichnen, wobei also für ein $ug \in ug \langle g', g \rangle$ gilt: $ug: E(g') \cup K(g') \xrightarrow{\text{in}} E(g) \cup K(g)$ und ug ist die Restriktion der Identität $\text{id} \in \text{id} \langle g, g \rangle$ auf $E(g') \cup K(g')$. Gilt $g' \underline{\text{tgg}}$, so beschreiben wir die daraus auf triviale Weise konstruierbaren Abbildungen mit $tg \langle g', g \rangle$.

Wir gehen nun über zu der Formulierung des obigen Sachverhalts bezüglich der Äquivalenzrelation: Genau dann soll g' *Teilgraph bis auf Äquivalenz von g* heißen (in Zeichen $g' \underline{\text{ätgg}}$), wenn ein $g'' \underline{\text{tgg}}$ existiert mit $g'' \underline{\text{ägg}}$. Analoges soll

für Untergraphen gelten: Wir nennen g' Untergraph bis auf Äquivalenz von g (in Zeichen: $g' \underline{\text{äug}} g$), g.d.w. es ein g'' so gibt, daß $g' \underline{\text{ägg}} g''$ und $g'' \underline{\text{ugg}} g$ ist.

Auch für $\underline{\text{äug}}$ bzw. $\underline{\text{ätg}}$ betrachten wir die entsprechenden, auf triviale Weise konstruierbaren Abbildungsmengen $\underline{\text{äug}}\langle g', g \rangle$ bzw. $\underline{\text{ätg}}\langle g', g \rangle$. Somit gibt es zu $\underline{\text{äug}} \in \underline{\text{äug}}\langle g', g \rangle$ ein $\underline{\text{äq}} \in \underline{\text{äq}}\langle g', g \rangle$ und ein $\underline{\text{ug}} \in \underline{\text{ug}}\langle g'', g \rangle$ mit $\underline{\text{äug}} = \underline{\text{äq}} \cdot \underline{\text{ug}}$. Analoges gilt für $\underline{\text{ätg}}$. Die Eigenschaften von $\underline{\text{äq}}$ und $\underline{\text{ug}}$ bzw. $\underline{\text{tg}}$ übertragen sich auf gewohnte Weise auf $\underline{\text{äug}}$ bzw. $\underline{\text{ätg}}$.

Es folgen noch einige Definitionen für die Benennung von Teilstrukturen in einem Graphen g :

Eine Kante $(e, e) \in K_r(g)$ heißt *Schlinge*. (Wir beschäftigen uns in dieser Arbeit der Einfachheit halber nur mit schlingenfreen Graphen.)

Sei $Z = \langle e_1, k_1, e_2, \dots, e_{n-1}, k_{n-1}, e_n \rangle$ eine nichtleere endliche Folge. Z heißt *ungerichteter Kantenzug* zwischen e_1 und e_n in g g.d.w. $e_1, \dots, e_n \in E(g)$, $k_1, \dots, k_{n-1} \in K(g)$ und $(\forall i \in \{1, \dots, n-1\}) ((ae(k_i) = e_i \wedge ee(k_i) = e_{i+1}) \vee (ee(k_i) = e_i \wedge ae(k_i) = e_{i+1}))$.

Wir bezeichnen einen ungerichteten Kantenzug Z zwischen e_1 und e_n in g , bei dem keine Ecke (und damit keine Kante) mehrfach vorkommt, mit *Weg* zwischen e_1 und e_n (in Zeichen: $\text{weg}(e_1, e_n)$). $AE(\text{weg}(e_1, e_n)) = e_1$ ist die *Anfangsecke* des Wegs zwischen e_1 und e_n , $EE(\text{weg}(e_1, e_n)) = e_n$ ist die *Endecke*.

Z heißt *gerichteter Kantenzug* zwischen e_1 und e_n in g , g.d.w.

Z ungerichteter Kantenzug zwischen e_1 und e_n in g ist und wenn für Z $(\forall i \in \{1, \dots, n-1\}) (ae(k_i) = e_i \wedge ee(k_i) = e_{i+1})$ gilt.

Gilt für Z $k_1, \dots, k_{n-1} \in K_m(g)$ und ist Z ein gerichteter Weg, so nennen wir $\text{weg}(e_1, e_n)$ einen *m-Pfad* in g zwischen e_1 und e_n (in Zeichen: $\text{pfad}(m, e_1, e_n)$). Noch ein Hinweis:

Wege und m-Pfade in g sind Teilgraphen von g und Ausdrücke wie $E(\text{weg}(e_1, e_n))$ sind somit sinnvoll. Die (möglicherweise leere) Menge der Wege bzw. m-Pfade zwischen e_1 und e_2 bezeichnen wir mit $\text{WEG}(e_1, e_2)$ bzw. $\text{Pfad}(m, e_1, e_2)$.

Trivialerweise ist $\text{WEG}(e_1, e_1) \neq \emptyset$, da - für $n=1$ -

von jeder Ecke zu sich selbst ein Weg Z (s.o.) existiert.

g besitzt Waldstruktur bzgl. r , g.d.w.

$(\forall e_1, e_2, e_3 \in E(g)) ((e_1, e_2), (e_1, e_3) \in K_r \rightarrow e_2 = e_3)$.

Besitzt g bzgl. r Waldstruktur, so nennen wir die Menge

$\text{Max}(r) := \{e \mid e \in E(g) \wedge (\forall e' \in E(g)) ((e, e') \notin K_r(g))\}$

die Menge der maximalen Elemente (Ecken) von g bzgl. r .

Ist $|\text{Max}(r)| = 1$, so besitzt g bzgl. r Baumstruktur. Die Menge

$\text{Blatt}(r) := \{e \mid e \in E(g) \wedge (\forall e' \in E(g)) ((e', e) \notin K_r(g))\}$ ist die Menge der Blätter in g bzgl. r .

Ist $g' \text{ ugg}$, dann heie $\text{con}(E(g) \setminus E(g'), K(g) \setminus K(g'))$ der Restgraph von g zu g' (in Zeichen: $\text{restgraph}(g, g')$).

2.2 HIERARCHISCHE GRAPHEN

Es ist unsere Absicht, Mechanismen anzugeben, mit denen man auch komplexe Manipulationen an zweidimensionalen Zeichenketten beschreiben kann. Wir wollen deshalb an dieser Stelle auf einen besonderen Graphentyp eingehen, der sich in seiner äußeren Erscheinung von dem in Kap. 2.1 unterscheidet, den hierarchischen Graphen. Diese sind schließlich auf die einfache Art von Graphen zurückführbar - von dieser Möglichkeit werden wir später noch Gebrauch machen -, wie es eben auch möglich ist, zweidimensionale Zeichenketten durch eindimensionale zu codieren *).

Hierarchische Graphen wurden geschaffen, weil es zur Darstellung mancher Sachverhalte notwendig erschien, den Ecken eines gewöhnlichen Graphen wiederum einen Graphen zuzuordnen, der der *Inhalt* der betreffenden Ecke genannt wird. Dies drücken wir dadurch aus, daß wir den Inhalt einer Ecke auch *in* diese hineinzeichnen.

Hier wird klar, weshalb in Figur 2.1/1 die Markierung einer Ecke in den Kreis, der die Ecke repräsentiert, gezeichnet wurde: Man kann den Inhalt als Markierung einer Ecke betrachten. Ist *cont* eine Funktion, die jeder Ecke einen Inhalt zuordnet, so gilt in Spezialfällen $\text{cont} = \text{emark}$.

Sei $\text{Graph}(\text{Eckm}, \text{Kam})$ die Menge aller Graphen über den Alphabeten *Eckm* und *Kam*. Ein *hierarchischer Graph* *h* (s.a. [51]) ist ein Graph $h = (E, \tilde{K} = \varphi)$ über $\text{Eckm} \cup \text{Graph}(\text{Eckm}, \text{Kam})$, bei dem

- 1) die Funktion $\text{emark}_h: E(h) \rightarrow \text{Eckm}$ zu einer Funktion *cont* erweitert wurde mit $\text{cont}: E(h) \rightarrow \text{Eckm} \cup \text{Graph}(\text{Eckm}, \text{Kam})$.

*) Es ist auch möglich, gewöhnliche Graphen als Spezialfälle von hierarchischen Graphen zu betrachten.

- 2) Für alle $g \in \text{Graph}(\text{Eckm}, \text{Kam})$ und $g = \text{cont}(e)$ mit $e \in E(h)$ gibt es eine mit * gekennzeichnete Ecke.
- 3) Es gibt eine mit * gekennzeichnete Ecke $e \in E(h)$, die *Startecke*⁺⁾ .

Die Inhaltsfunktion *cont* ordnet jeder Ecke als Markierung ein Element aus *Eckm* (bei hierarchischen Graphen auch *primitives Element* genannt) oder einen Graphen über *Eckm* und *Kam* zu.

Die Kantenmenge (zwischen den Ecken von *h*!) ist leer. Wichtig ist der Hinweis, daß bei hierarchischen Graphen i. a. $E(h) \cap \text{Eckm} \neq \emptyset$ ist; dies ist bei 'sinnvollen' hierarchischen Graphen sogar die Regel.

Betrachten wir in Figur 2.2/2 ein Beispiel für einen hierarchischen Graphen bei dem

$E = \{\text{var1}, \text{var2}, \text{const}, \text{akt1}, \text{akt2}, \text{abfr}, \text{proz1}, \text{proz2}, \text{hauptprogramm}\}$,
 $\text{Eckm} = \{0, 1, \text{abfr}, \text{akt1}, \text{akt2}, \text{proz1}, \text{proz2}, \text{var1} := \text{const}, \text{var2} := \text{var1}, \text{var1} > 0\}$,

$\text{Kam} = \{0, 1\}$

und *cont* durch Figur 2.2/1 definiert wird.

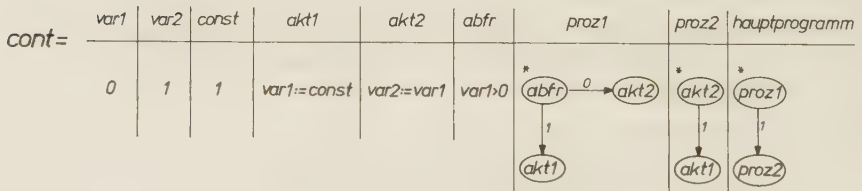


Fig. 2.2/1

⁺⁾ Die Bedeutung der Kennzeichnung von Ecken durch * ist für uns nicht weiter interessant. Deshalb nehmen wir die ausgezeichnete Ecke nicht mit in die Tupeldarstellung für *h*.

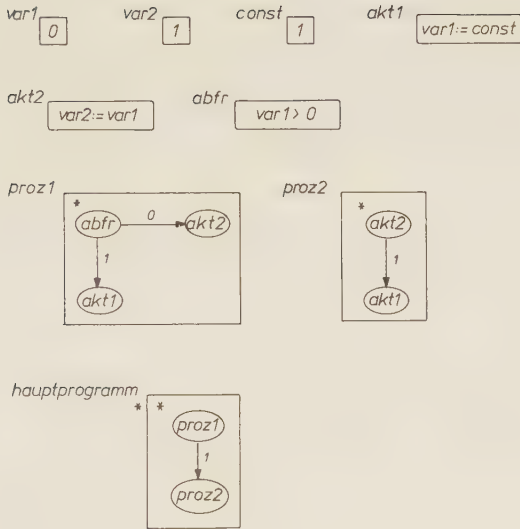


Fig. 2.2/2

Bisher haben wir für hierarchische Graphen nur eine abstrakte *Definition* und in exemplarischer Form eine graphische *Repräsentierung* für Objekte angegeben, die die Definition erfüllen. Was noch fehlt, ist eine Angabe darüber, wie diese Zeichenketten zu interpretieren sind, eben die *Semantik* von hierarchischen Graphen. Da sie für uns aber nicht interessant ist, soll dies nur grob geschehen, jedoch darauf hingewiesen werden, daß diese dreiteilige Einführung von Sachverhalten künftig wiederholt auftreten wird.

Die Bezeichnungen in Figur 2.2/2 sind bereits so suggestiv gewählt um zu zeigen, daß hierarchische Graphen zur Beschreibung von Programmläufen dienen, also selbst zur Darstellung der Semantik von Programmen verwendet werden können. Der Inhalt einer Ecke entspricht analog zur ALGOL 68-Terminologie dem *Wert eines Objekts*; speziell bei Prozeduren besitzt der Prozedurname den Prozedurkörper als Wert^{*)}.

Halten wir fest: Hierarchische Graphen sind der erste Schritt auf einem Weg, der von Graphen mit 'einfachen' Eckenmarkierungen zu Graphen mit 'komplexen' Eckenmarkierungen führt.

Abschließend zu diesen Ausführungen über hierarchische Graphen soll noch an einem Beispiel der Zusammenhang zwischen formaler Definition, graphischer Repräsentierung und Interpretation (Semantik) dargestellt werden: Lassen wir zu, daß in der formalen Definition die Markierung einer Kante nicht nur mit Elementen aus dem Alphabet Kam erlaubt sein soll, sondern auch aus der potentiell unendlichen Menge $Kam \cup Graph(Eckm, Kam)$! In der graphischen Repräsentierung ändert sich nichts. Auch diese Markierungen werden einfach an die Kanten geschrieben. Wie sollen nun die neuen Objekte zu interpretieren sein? Im programmiersprachlichen Bereich läßt sich nicht leicht dazu eine sinnvolle Interpretation finden. In [14] jedoch werden 'graph expressions' definiert, die sich als hierarchische Graphen darstellen lassen, wenn man Graphen als Kantenmarkierungen verwendet. In diesem Zusammenhang etwa würde die Kantenmarkierung angeben, wie diese Kante in einem Ableitungsschritt überzuführen wäre. Nicht alle hierarchischen Graphen sind aber 'sinnvolle', also interpretierbare 'graph expressions'.

^{*)} Dies sollte nicht mit dem *Ergebnis* einer Prozedur verwechselt werden.

Definition, Repräsentierung und Interpretation mathematischer Objekte sind also miteinander korreliert, wodurch sich weitere Entwurfsschwierigkeiten ergeben, denn Objekte, die die Definition erfüllen, sollen in jedem Fall repräsentierbar (und interpretierbar) sein. Diese Problematik tritt ganz besonders im Kap. 6 auf. Sie sollte aber bereits an dieser Stelle potentiellen Benutzern des im folgenden entwickelten Systems nähergebracht werden.

3 QUASIHIERARCHISCHE GRAPHEN

Es ist unser Ziel, Methoden anzugeben, die es gestatten, Veränderungen an zweidimensionalen Zeichenketten formal beschreiben zu können. Die Graphen dienen wiederum zur Beschreibung von anderen Sachverhalten. Um letztere klar darzustellen, erweisen sich in einigen Fällen gewöhnliche Graphen als zu wenig suggestiv. Deshalb sind wir zur Definition hierarchischer Graphen übergegangen. Viele in der Literatur angegebenen graphischen Darstellungen lassen sich weder mit gewöhnlichen Graphen noch mit hierarchischen Graphen formal erfassen. Es sind solche, bei denen in den Ecken der Darstellungen noch Graphen zu finden sind, in deren Ecken wiederum Graphen stehen u.s.w.. Obendrein verlaufen Kanten zwischen Ecken beliebiger Hierarchiestufe. Wir versuchen solche graphischen Darstellungen zu formalisieren, indem wir eine weitere Modifizierung am Graphbegriff vornehmen und die *quasihierarchischen Graphen* einführen.

Ein Beispiel für einen quasihierarchischen Graphen ist in Figur 3/2 angegeben. Hier sieht man sofort, daß es für die Definition von quasihierarchischen Graphen nicht genügt, auf einfache Weise den Wertebereich der Funktion *cont* auf die Menge aller hierarchischen Graphen auszuweiten. Es kann auf befriedigende Weise nicht motiviert werden, ob man zum Inhalt der Ecke 4 in Figur 3/2 die 'Stummel' der in die Ecken 5 bzw. 6 einlaufenden Kanten noch hinzunehmen soll oder nicht. Die Inhaltsfunktion soll im folgenden mit Hilfe einer neuen Markierung in dargestellt werden. Dadurch ist es möglich, die graphischen Repräsentierungen gegebenenfalls leicht zu 'enthierarchisieren' *).

*) Der besonderen Bedeutung von in wird in Kap. 6 Rechnung getragen.

Definition 3/:

Ein *quasihierarchischer Graph* q ist ein Graph über den Alphabeten $Eck \cup \{\varepsilon\}$ und $Kam \cup \{\underline{in}\}$, so daß gilt

- 1) q besitzt bezüglich $\underline{in}$ Waldstruktur
- 2) $(\forall k \in K_{\underline{in}}(q)) (emark(ee(k)) = \cdot)$. $\blacktriangle$

Auch gewöhnliche hierarchische Graphen h lassen sich leicht durch diese Definition beschreiben. Dabei wird $cont$ dargestellt durch:

$$(\forall e \in E(h)) (e' \in E(cont(e)) \Rightarrow (e', e) \in K_{\underline{in}}(h)).$$

Für die graphische Repräsentierung der $\underline{in}$ -Relation bieten sich zwei Möglichkeiten an, eine *explizite* und eine *implizite*. Bei der expliziten Darstellung tritt die Markierung $\underline{in}$ direkt auf, bei der impliziten wird die Ecke e_1 in die Ecke e_2 gezeichnet, wenn $(e_1, e_2) \in K_{\underline{in}}$ ist. Dies führt zu einer größeren Übersichtlichkeit der graphischen Darstellung.

Demnach sollen die Darstellungen in Figur 3/1 und 3/2 denselben Graphen repräsentieren.

quasihierarchischer Graph in expliziter Darstellung

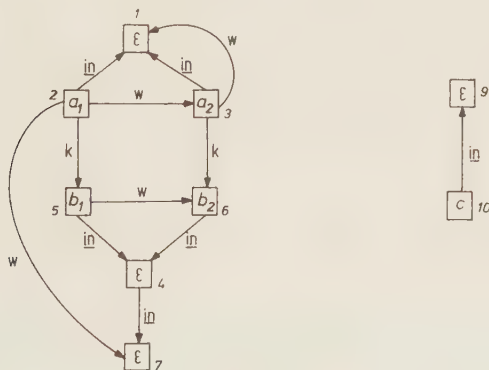


Fig. 3/1

quasihierarchischer Graph in impliziter Darstellung

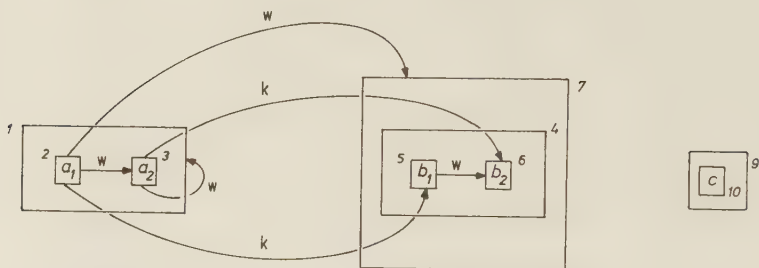


Fig. 3/2

Voraussetzung für jede Manipulation von Zeichenketten ist eine genaue Kenntnis darüber, wie eine Zeichenkette zu zerlegen ist, d. h. eine Angabe darüber, aus welchen *atomaren*, also nicht weiter zerlegbaren Zeichen die Zeichenkette besteht. Erst dann ist der Begriff *Kontext* eines Zeichens, also das Gefüge, in dem das betreffende Zeichen steht, sinnvoll. Bei linear geordneten Zeichenketten, z. B. Buchstabenfolgen, ist dies einfach, bei Graphen bietet sich eine Zerlegung in die Menge der Untergraphen an, bei hierarchischen Graphen könnte man die Ecken bzw. deren Inhalte als manipulierbare Zeichen betrachten, bei quasihierarchischen Graphen ist es nicht so leicht, eine sinnvolle (was auch dies immer heißen mag) Zerlegung in atomare Zeichen zu geben. Wie man die implizit dargestellten quasihierarchischen Graphen als solche formal manipulieren kann, wird an anderer Stelle noch diskutiert.

Wie können wir uns aber fürs erste aus dem Dilemma be-

freien, daß wir einerseits quasihierarchische Graphen als Beschreibungsmittel für programmiersprachliche Sachverhalte verwenden wollen, ihre Manipulation aber über gewöhnliche Graphen zu beschreiben beabsichtigen?

Ein gangbarer Weg ist die Angabe einer Übersetzungsvorschrift, die definiert, wie man die Zeichenketten für quasihierarchische Graphen in der impliziten Form in solche der expliziten Darstellungsweise übersetzen kann und umgekehrt. Die Existenz einer solchen Übersetzungsvorschrift können wir aber voraussetzen.

In diesem Kapitel über quasihierarchische Graphen soll noch auf eine weitere Schwierigkeit eingegangen werden, die sich beim Übergang zur Klasse der äquivalenten quasihierarchischen Graphen im Sinne von Kap. 2.1 bei Verwendung der impliziten Darstellung ergibt. Um auf die Relation $\underline{ag}$ zu kommen, definierten wir uns einen Isomorphismus h zwischen den Eckenmengen zweier Graphen und forderten: Wenn eine Kante mit der Markierung r zwischen zwei Ecken verläuft, dann muß auch eine solche Kante zwischen den Bildecken verlaufen. Zusätzlich mußte $emark$ für die Ecken und die Bildecken dieselben Werte liefern. Mit der Einführung dieses Isomorphismus wollten wir die Eckenbezeichnungen vernachlässigen können. Wenn nun in der impliziten Darstellung einem Knoten ein ganzer Graph als Markierung zugeordnet wird (wie es bei den hierarchischen Graphen geschieht), so gehören also auch die Eckenbezeichnungen mit zur Markierung. Der o. g. Isomorphismus h wäre somit auf die Eckenbezeichnungen der Graphen, die als Markierung dienen, auszudehnen. Dies soll aber an dieser Stelle nicht geschehen.

Um Übersicht zu behalten, muß man sich zur Darstellung unseres Systems zunächst auf die Manipulation gewöhnlicher Graphen beschränken, um nicht noch die Schwierigkeiten, die sich durch die Einführung des komplexeren

Zeichenkettensystems der (impliziten) quasihierarchischen Graphen ergeben, mitführen zu müssen.

Es soll noch darauf hingewiesen werden, daß wir in diesem Zusammenhang zum ersten Mal mit dem Problem der *Semantik* von Zeichenketten konfrontiert werden. Wir werden an geeigneter Stelle noch genauer darauf eingehen. Doch soviel sei vorweggenommen: Wir definieren die Semantik einer Zeichenkette, wenn wir angeben, wie man von einer gegebenen Zeichenkette z ausgehend - z. B. von einem quasihierarchischen Graphen in impliziter Form - eine der Zeichenkette z zugeordnete Zeichenkette z' findet - z. B. ein quasihierarchischer Graph in der expliziten Form - (von der man annimmt, man kann sie 'besser verstehen').

4 GRAPHMANIPULATIONSVORSCHRIFTEN

In vorangehenden Kapiteln sprachen wir von Graphen und Ersetzungen von Teilen daraus durch andere. Dabei wurde uns auch das Problem bewußt, daß der Kontext, innerhalb dessen ein ersetzbarer Graph oder eine Zeichenkette steht, von großer Bedeutung ist. Man sieht sofort, daß Fragestellungen etwa der Gestalt auftreten: Ersetze die vorgegebene Zeichenkette^{*)} durch eine andere unter der Voraussetzung, daß eine dritte Zeichenkette vorliegt (oder nicht vorliegt); bette die neue Zeichenkette nach vorgegebenem Schema in die Restzeichenkette ein, die verbleibt, wenn man aus der Wirtszeichenkette die zu ersetzende Zeichenkette entfernt hat.

Wir wollen diese Mechanismen für Graphen definieren. Hilfreich für das Verständnis ist, sich das Schema der Figur 4/1 vor Augen zu halten. In ihm werden durch den metalinguistischen Konnektor Υ die 3 Komponenten 'zu ersetzende Zeichenkette', 'neu einzusetzende Zeichenkette' und 'Kontext' integriert^{**)}.

Im folgenden sind die Eckenbezeichnungen der behandelten Graphen meist irrelevant. Deshalb gehen wir manchmal über zu Äquivalenzklassen von Graphen über den Alphabeten Eckm und Kam bzgl. der in Kap. 2.1 eingeführten Relation $\dot{a}g$ und arbeiten nur mit Klassen und Repräsentanten. Da sich keine Mehrdeutigkeiten ergeben, bleiben wir beim Begriff Graph und vermeiden eine neue Bezeichnung für die Repräsentanten (etwa: *Diagramm*).

*) Im folgenden seien mit Zeichenketten auch zweidimensionale Gebilde gemeint.

**) In der Backus-Naur-Form (BNF) entspricht Υ dem $::=$ -Zeichen, die 'zu ersetzende Zeichenkette' ist die linke Seite der Produktion, die 'neu einzusetzende Zeichenkette' entspricht der rechten Seite. Der Kontextbegriff fehlt natürlich in der BNF. (Wir betrachten nur BNF-Regeln, die keine Alternativen auf der rechten Seite besitzen.)

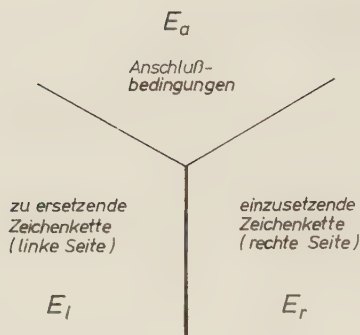


Fig. 4/1

Definition 4/1:

Ein Graph f heie *Manipulationsvorschrift*, g.d.w.

- 1) $E(f) = E_r(f) \cup E_l(f) \cup E_a(f)$,
- 2) 2.1) $K(f) \cap (E_l(f) \times E_r(f)) = \emptyset$,
 2.2) $K(f) \cap (E_r(f) \times E_l(f)) = \emptyset$,
- 3) $|E_l(f)| \geq 1$. ▲

Der Untergraph $\text{con}(E_l(f), (E_l(f) \times E_l(f)) \cap K(f)) = \text{con}(E_l(f), K(f))$ heie die *linke Seite* von f (in Zeichen: $\text{liS}(f)$) und analog dazu $\text{con}(E_r(f), (E_r(f) \times E_r(f)) \cap K(f)) = \text{con}(E_r(f), K(f))$ die *rechte Seite* von f (in Zeichen: $\text{reS}(f)$).

Die Ecken von f werden also in drei Bereiche aufgeteilt, bei denen zumindest E_l nicht leer ist. Wichtig ist die Einschränkung, da zwischen E_l und E_r keine Kanten verlaufen drfen; zwischen den anderen Bereichen knnen beliebige Verbindungen bestehen. Jedes f lt sich im allgemeinen auf mehrfache Art als Manipulationsvorschrift darstellen. Speziell lt sich der Graph (a) nur auf genau eine Weise als Manipulationsvorschrift auffassen (siehe Figur 4/2), der Graph (a) $\times$ a hingegen auf genau drei Arten (siehe Figur 4/3(a-c)).

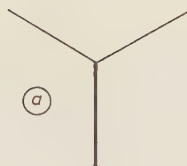


Fig. 4/2

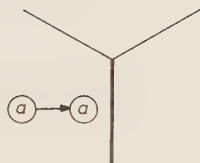


Fig. 4/3(a)



Fig. 4/3(b)

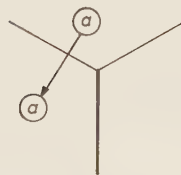


Fig. 4/3(c)

Definition 4/2:

Sei f eine Manipulationsvorschrift. Es heißen

- 1) $aK(f) := K(f) \cap ((E_l(f) \times E_a(f)) \cup (E_a(f) \times E_l(f)))$ die *alten Kanten von f* und
- 2) $nK(f) := K(f) \cap ((E_r(f) \times E_a(f)) \cup (E_a(f) \times E_r(f)))$ die *neuen Kanten von f* . ▲

Die durch 1) beschriebenen Kanten von f sind also jene, die vom 'linken' Ast des Konnektors $\vee$ geschnitten werden und die von 2) werden vom 'rechten' geschnitten.

Um klarzustellen, wie wir Manipulationsvorschriften verwenden werden, benötigen wir noch einige Begriffe:

Definition 4/3:

Sei f eine Manipulationsvorschrift und $k \in K(f)$. $A(k)$ heißt die *Anschlußbecke* von k , wobei gilt: $A(k) := ae(k)$, wenn $ae(k) \in E_a(f)$ oder $A(k) := ee(k)$, wenn $ee(k) \in E_a(f)$. Δ

Anschlußbecken sind somit jene Ecken aus $E_a(f)$, die durch eine Kante mit $E_r(f)$ verbunden sind.

Die Figur 4/4 gibt Beispiele für die bisherigen Definitionen.

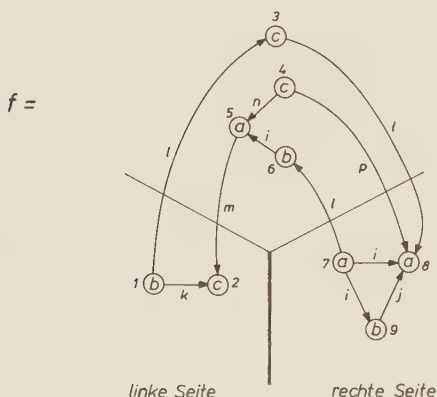


Fig. 4/4

Die Ecke 3 ist die Anschlußbecke der Kante $(3,8)$, $A((4,8))=4$, $A((7,6))=6$. Für die neuen Kanten gilt $nK(f) = \{(7,6), (4,8), (3,8)\}$, für die alten $aK(f) = \{(1,3), (5,2)\}$.

Definition 4/4:

Sei f eine Manipulationsvorschrift und $k \in nK(f)$. Mit Anschlußbedingung für k (in Zeichen: $\text{anb}(k)$) werde jener Teilgraph ttgf benannt, für den gilt:

- 1) $E(t) = \{e \mid e \in E_1(f) \cup E_a(f) \wedge (\exists w \in \text{WEG}(A(k), e)) (E(w) \cap E_r(f) = \emptyset \wedge (e' \in E(w) \cap E_1(f) \rightarrow e' = EE(w)))\}$
- 2) $K(t) = (K(f) \cap (E(t) \times E(t))) \setminus K(\text{liS}(f))$. ▲

$E(t)$ besteht aus allen Ecken von f , die mit k über einen Weg verbunden sind. Die Wege führen nicht über $E_r(f)$, und nur jene Ecken e' aus $E_1(f)$ werden zu $E(t)$ hinzugenommen, die höchstens Endecke eines Wegs von $A(k)$ nach e' sind. (Also ist e' entweder $ae(k')$ oder $ee(k')$ für $k' \in aK(f)$). $K(t)$ beinhaltet keine Kante aus $\text{liS}(f)$. Insofern ist t i. a. nur Teilgraph von f .

Die Figur 4/5 erläutert Definition 4/4 anschaulich, wobei der gezackt umrandete Bereich die Anschlußbedingung $\text{anb}((e', e''))$ darstellt.

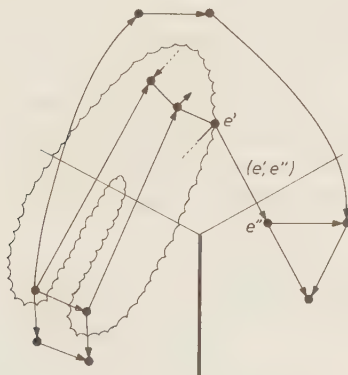


Fig. 4/5

Bisher haben wir Manipulationsvorschriften definiert und Teile daraus gekennzeichnet. Diesem Vorgehen entspräche in der Theorie der Formalen Sprachen die Einführung des Produktionsbegriffs. Was uns noch fehlt ist ein Analogon zum Ableitungsbegriff, im folgenden Anwendung einer Manipulationsvorschrift f bezüglich eines Graphen g genannt.

Eine Manipulationsvorschrift ist zunächst nur ein Graph. Erst im Zusammenhang mit einem weiteren Graphen kommt ihr eine Bedeutung zu. Dies ist der dritte Schritt im Vorgehen, zunächst eine formale Definition eines mathematischen Objektes anzugeben, dann die graphische Repräsentierung und schließlich eine Interpretation.

Etwas präziser als am Anfang des Paragraphen können wir nun angeben, wie eine Manipulationsvorschrift f bezüglich eines Graphen g zu interpretieren ist: Der Graph der linken Seite des Konnektors $\vee$ von f wird in g als Untergraph (bis auf Äquivalenz) aufgesucht, dieser Anteil aus g entnommen und dafür die rechte Seite von $\vee$ eingesetzt. Für jedes $k \in K(f)$ wird $\text{anb}(k)$ als Teilgraph (bis auf Äquivalenz) im Restgraphen aufgesucht^{*)}. $\text{anb}(k)$ findet sich i. a. an mehreren Stellen im Restgraphen. Für jedes Auftreten wird im Restgraphen die zur Anschlußecke von k korrespondierende Ecke bestimmt und zwischen dieser und der entsprechenden Ecke des eingesetzten Graphen eine Kante mit gleicher Richtung und Markierung eingetragen. Für die Definition des Ersetzungsvorganges benötigen wir noch einige Begriffe.

Definition 4/5:

Seien t , u und g Graphen, die folgende Bedingungen erfüllen:

*) Diese Aufteilung zwischen dem, was Teil- und was Untergraph sein soll, ist zunächst willkürlich und wird im folgenden noch motiviert.

1) $\underline{u}gg$

2) $\underline{t}gg$

3) $E(g) = E(t) \cup E(u)$

Ist $H := ((E(u) \times E(t)) \cup (E(t) \times E(u))) \cap K(g)$, dann heit u der Untergraphanteil von g und $T(u, g) := \text{con}(E(t) \cup \{e | e \in E(u) \wedge \exists (k \in H) (e = ae(k) \vee e = ee(k))\}, K(g))$ Teilgraphanteil von g^* . ▲

Die Figur 4/6 erlutert die Definition 4/5 (Kantenmarkierungen wurden weggelassen).

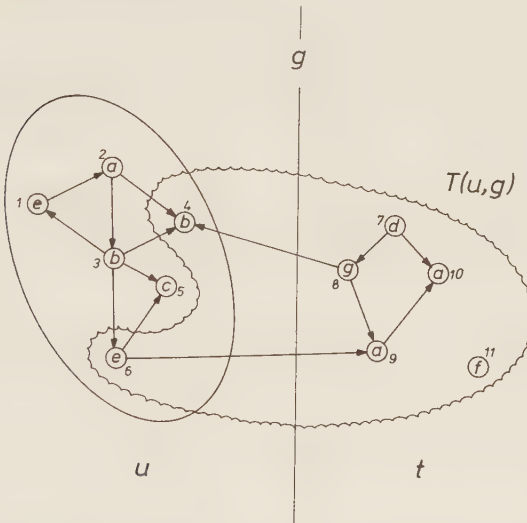


Fig. 4/6

Fr $k \in K(f)$ einer Manipulationsvorschrift f suchen wir $ANB(k)$ als quivalenten Unterteilgraph in einem Graphen auf. Dabei ist der Untergraphanteil von $ANB(k)$ $liS(f)$ und der Teilgraphanteil $T(liS(f), ANB(k)) = anb(k)$.

*) Diese Bezeichnungen werden im Hinblick auf Definition 4/6 und vor allem auf Definition 4/8 gewhlt.

Definition 4/6:

Seien u , g und h Graphen, mit $u \subseteq g$. g heie *äquivalenter Unterteilgraph* von h (in Zeichen: $g \dot{\subseteq} h$) g.d.w.

1) $g \dot{\subseteq} h$

2) $u \dot{\subseteq} h$

3) $(\exists \dot{\subseteq} g \subseteq \dot{\subseteq} h) (\exists \dot{\subseteq} g \subseteq \dot{\subseteq} h) (\dot{\subseteq} g = \text{restrikt}(\dot{\subseteq} h, u))$.

Die Menge der aus der Gültigkeit von $g \dot{\subseteq} h$ konstruierbaren Graphhomomorphismen bezeichnen wir mit $\dot{\subseteq} h, g$. Δ

Der Graph g mu also in seiner Gesamtheit als Teilgraph in h liegen und der vorgegebene Untergraph u von g als Untergraph in h . Da u auch an anderen Stellen in h liegen kann, als an der von $\dot{\subseteq} h$ bestimmten, 'fixiert' Bedingung 3) u auf den betrachteten Bereich.

Wir verwenden die in Definition 4/5 und 4/6 eingeführten Begriffe für Manipulationsvorschriften und bilden für $k \in K(f)$ einen Graphen, der aus $\text{anb}(k)$ und $\text{liS}(f)$ besteht.

Definition 4/7:

Sei f eine Manipulationsvorschrift und $k \in K(f)$. Wir nennen den Graphen

$$\text{ANB}(k) = \text{con}(\text{E}_1(f) \cup \text{E}(\text{anb}(k)), K(f))$$

die *Gesamtanschlußbedingung* für k . Δ

Definition 4/8:

Es sei g , g' , l und der einer Manipulationsvorschrift f zugrunde liegende Graph Graphen über Eckm und Kam mit $E(g) \cap E(f) = \emptyset$, $l \dot{\subseteq} \text{liS}(f)$, $l \subseteq g$ und $m \in \text{Kam}$. Für jedes $k \in K(f)$ und für $\lambda \in \dot{\subseteq} \text{liS}(f), l$ bestimmen wir

$$F(k) := \{\varphi_k \mid \varphi_k \in \dot{\subseteq} \text{ANB}(k), g \wedge \text{restrikt}(\varphi_k, \text{liS}(f)) = \lambda\}^*.$$

Der Graph g' heie das Ergebnis der Anwendung von f auf g in λ (in Zeichen: $\text{anwend}(f, g, \lambda)$) g.d.w.

$$g' = \text{con}(E(\text{restgraph}(g, l)) \cup E(\text{reS}(f)), K(\text{restgraph}(g, l)) \cup$$

$$K(\text{reS}(f)) \cup \bigcup_{k \in K(f)} \text{Neu}(k)), \text{ wobei}$$

*) Alle φ_k , von denen es bei festem k mehrere geben kann, liefern bezgl. ihrer Restriktion auf die linke Seite von f dieselben Werte.

$$\text{Neu}(k) := \{k' \mid (\exists \varphi_k \in F(k)) ((k = (A(k), ee(k), m) \rightarrow k' = (\varphi_k(A(k)), ee(k), m)) \wedge (k = (ae(k), A(k), m) \rightarrow k' = (ae(k), \varphi_k(A(k)), m)))\}.$$

Wir vereinbaren noch, daß auch $\text{anwend}(f, g, l)$ geschrieben werden darf, wenn $|\text{äq} < \text{liS}(f), l| = 1$ oder nur $\text{anwend}(f, g)$, wenn genau ein lugg mit $\text{läqliS}(f)$ existiert. Zusätzlich vereinbaren wir, daß $g' = g$ ist, wenn nicht $\text{liS}(f) \text{äugg}$ ist. ▲

Trivialerweise ist $\text{reS}(f) \text{ugg}'$ und $\text{restgraph}(g, l) = \text{restgraph}(g', \text{reS}(f))$.

Die Definition 4/8 beschreibt einen wichtigen, zentralen Begriff in dieser Arbeit. Deshalb wollen wir ihn ausführlich anhand von Beispielen erläutern. Dazu benötigen wir noch einen Begriff.

Definition 4/9:

Sei f eine Manipulationsvorschrift und $k \in K(f)$. Die Menge $E_1(f) \cap E(\text{anb}(k))$ heiße die *Ankereckenmenge* von k (in Zeichen: $\text{anker}(k)$). ▲

Satz 4/:

Seien $f, g, k, l, \varphi_k, \lambda$ und $F(k)$ wie in Definition 4/8 vereinbart. Dann gilt:

$$\lambda(\text{anker}(k)) = \bigcup_{\varphi_k \in F(k)} \varphi_k(E(\text{anb}(k))). \quad \blacktriangle$$

Beweis:

Die Behauptung folgt unmittelbar aus den Definitionen 4/7, 4/8 und 4/9. ▲

Von den Bildern der Ankerecken unter λ ausgehend, wird für $k \in K(f)$ der Graph $\text{anb}(k)$ als (äquivalenter) Teilgraph in $\text{restgraph}(g, l)$ gesucht. Hier ist der Suchvorgang 'verankert'. Durch die Angabe von λ , und damit des Untergraphen von g , der ersetzt werden soll, ist für $\text{ANB}(k)$ schließlich nur noch $\text{anb}(k)$ relevant.

Beispiel 4/1:

Wir wollen zunächst an einem *abstrakten* Beispiel den Sachverhalt für $\text{anwend}(f, g, \lambda)$ klären. Figur 4/7(a) beschreibt f . Anhand geometrischer Figuren wird dargestellt, in welche Bereiche der f zugrunde liegende Graph zerfällt: Das Quadrat veranschaulicht die linke Seite, das Dreieck die rechte. Der Einfachheit halber wird angenommen, daß die Kanten der in diesem Beispiel beteiligten Graphen keine Markierungen tragen sollen. Die neuen Kanten von f sind verschieden dargestellt, um ihre Gesamtanschlußbedingungen einfacher erkennen zu können. Die Figuren im oberen Bereich des Konnektors λ beschreiben die Teile des f zugrunde liegenden Graphen, die die Anschlußbedingungen der neuen Kanten von f repräsentieren. Die Figur 4/7(b) stelle g dar. In Figur 4/7(c) sind die zu den Ankerecken *aller* neuen Kanten von f korrespondierenden Ecken von g gezeichnet. In den Figuren 4/7(d-g) wird der für die betrachtete neue Kante von f relevante Teil von g hervorgehoben und mit 'hilfsgraph' bezeichnet. Die für jedes Auffinden von $\text{anb}(k)$ auftretenden Bilder der Anschlußecken $A(k)$ sind in hilfsgraph ebenfalls eingetragen. Figur 4/7(g) beschreibt den Sonderfall einer leeren Ankereckenmenge, Figur 4/7(h) den Graphen, der aus den noch nicht verbundenen Teilen $\text{restgraph}(g, l)$ und $\text{reS}(f)$ besteht. Die Figuren 4/7(i-l) korrespondieren zu $\text{Neu}(k)$ der Definition 4/8. In ihnen wird der für die betrachtete neue Kante von f relevante Teil von g hervorgehoben und mit 'neugraph' bezeichnet. Figur 4/7(m) zeigt g als Ganzes.

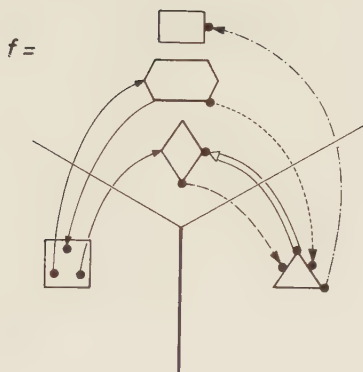


Fig. 4/7(a)

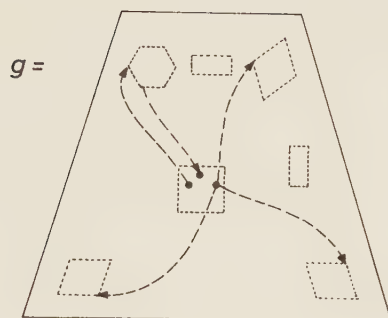


Fig. 4/7(b)

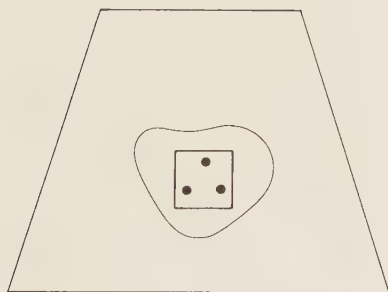


Fig. 4/7(c)

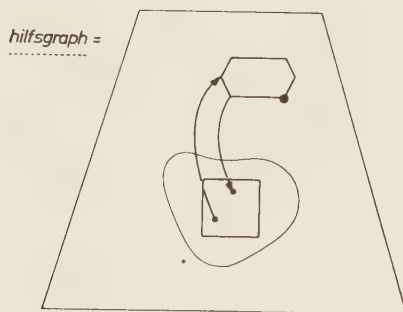


Fig. 4/7(d)

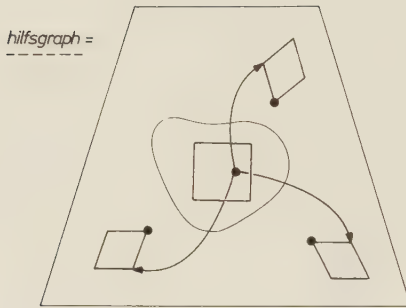


Fig. 4/7(e)

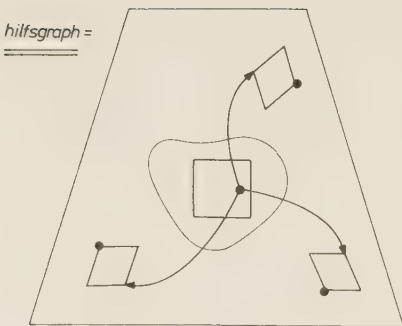


Fig. 4/7(f)

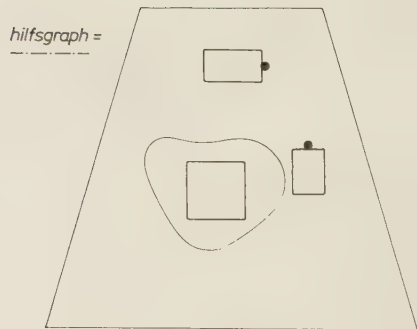


Fig. 4/7(g)

$$\text{con}(E(\text{restgraph}(g,l) \cup E(\text{reS}(f)), \\ K(\text{restgraph}(g,l) \cup K(\text{reS}(f)))) =$$

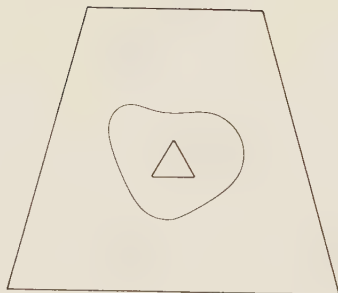


Fig. 4/7(h)

neugraph =

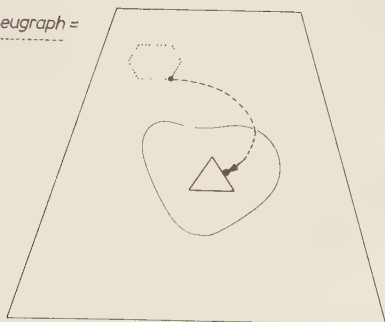


Fig. 4/7(i)

neugraph =

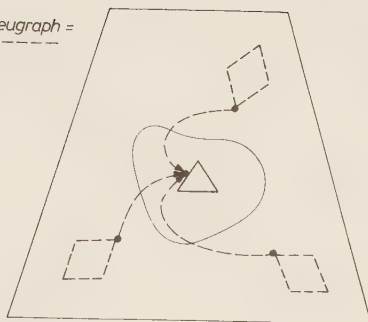


Fig. 4/7(lj)

neugraph =

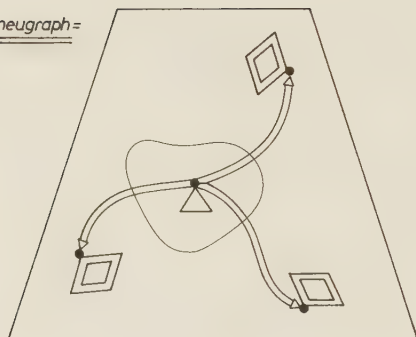


Fig. 4/7(k)

neugraph =

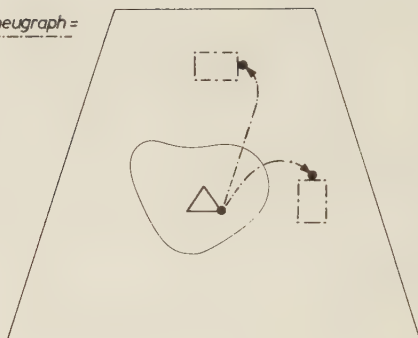


Fig. 4/7(l)

$g' =$

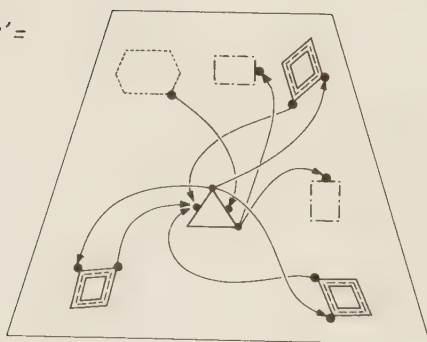


Fig. 4/7(m)

Das folgende Beispiel macht deutlich, daß die Suche nach Teilgraphen in einem Graphen u. U. recht umfangreich werden kann.

Beispiel 4/2:

Gegeben sei die folgende Manipulationsvorschrift f und ein Graph g , auf den sie angewendet wird. Die linke Seite von f und die Ecke von g , die Untergraph von g ist und äquivalent zu $liS(f)$, werde der Einfachheit halber durch eine Kreislinie gekennzeichnet. Die Ziffern an den Kanten von g sollen unterscheiden, in welchen Kombinationen die (einzige) Anschlußbedingung in g auftritt: Je drei Kanten, an denen die gleiche Ziffer auftaucht, bilden mit den Ecken einen zur Gesamtanschlußbedingung äquivalenten Teilgraph von g (s. Figuren 4/8(a-b)).

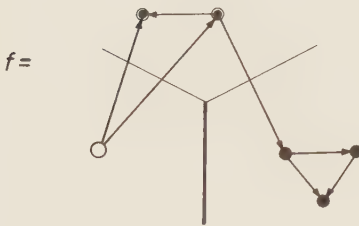


Fig. 4/8(a)

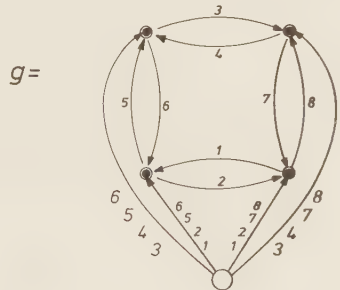


Fig. 4/8(b)



Im Beispiel 4/1 wurden einige Sonderfälle noch nicht berücksichtigt, die bei der Verwendung eines Graphen als Manipulationsvorschrift f auftreten können. Diese Fälle werden hier gleich in der 1-Darstellung abgehandelt:

- 1) Ein Untergraph u von f wird dem oberen Teil von i zugeordnet ohne Verbindung mit $liS(f)$ und $reS(f)$, siehe dazu Figur 4/9.

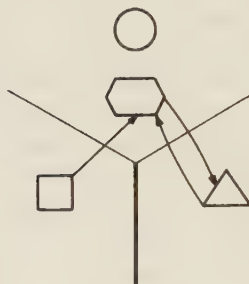


Fig. 4/9

Dieser Anteil, durch einen Kreis symbolisiert, ist laut Definition 4/8 irrelevant, da er für kein $k \in K(f)$ in $ANB(k)$ auftritt^{*)}.

- 2) Ein Untergraph u von f wird dem oberen Teil von γ zugeordnet, hat Verbindung mit $liS(f)$, aber nicht zu $reS(f)$, siehe Figur 4/10. Auch dieser Anteil von f ist für die Operation anwend irrelevant, da er ebenfalls für kein $k \in K(f)$ in $ANB(k)$ auftritt^{*)}.

^{*)} Somit kann $anwend(f_1, g, l) = angewend(f_2, g, l)$ auf triviale Weise trotz $f_1 \neq f_2$ gelten.

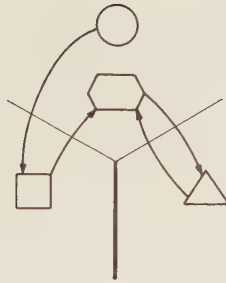


Fig. 4/10

- 3) In Definition 4/1 forderten wir, daß bei Manipulationsvorschriften f $|E_1(f)| \geq 1$ sein muß. Diese an sich nicht unbedingt notwendige Bedingung bewirkt, daß nicht bei leerer linker Seite von f die Manipulationsvorschrift auf jeden Graphen anwendbar ist, da der 'leere Graph' Untergraph jedes beliebigen Graphen ist. Dies wäre gleichbedeutend mit einem nichtdeterministischen Auftauchen einer rechten Seite in einem Graphen g auf den f angewendet wurde.
- 4) Die Figur 4/11 beschreibt den Fall, daß $E_a(f)$ leer ist. Angenommen es gilt $liS(f) \text{ äugg}$ für ein g .

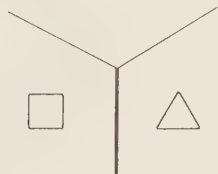


Fig. 4/11

g wird durch die Figur 4/12(b) beschrieben, bei der durch einen Kreisbogen der Untergraph abgetrennt ist, der ersetzt werden soll und somit äquivalent zur $liS(f)$ ist. Die zu $anb(\dots\dots\rangle)$ bzw. $anb(-----\rangle)$ äquivalenten Teilgraphen und die zu den Anschlußecken korrespondierenden Ecken sind gekennzeichnet.



Fig. 4/12(b)

*) Man beachte, daß hier auch ein zu $anb(\dots\dots\rangle)$ äquivalenter Teilgraph von g gefunden werden kann, aber $anb(\dots\dots\rangle)$ hat Ankerecken und muß somit Verbindung zu dem Teil von g besitzen, der äquivalent zur $liS(f)$ ist.

$\text{anwend}(f,g) =$

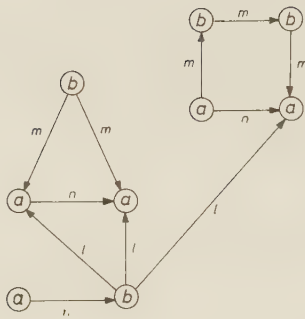


Fig. 4/13(c)



Nach diesen Beispielen, die die Funktionsweise der Definition 4/8 veranschaulichen sollten, diskutieren wir noch kurz andere durchaus sinnvolle Möglichkeiten, wie man einen Graphen f auf einen Graphen g anwenden kann, kurzum, wir entwickeln einige Alternativen für die Definition von $\text{anwend}(f,g,\lambda)$ bzw. $\text{anwend}(f,g,l)$.

- 1) Ist l vorgegeben und $\text{liS}(f) \underline{\text{ägl}}$, so hat anwend die Wirkung, daß l aus g entfernt wird und $\text{reS}(f)$ eingesetzt wird. Es muß für kein $k \in K(f)$ $\text{ANB}(k) \underline{\text{äutgg}}$ sein, also speziell $\text{anb}(k)$ muß nicht in g als Teilgraph auffindbar sein. Es ist denkbar, anwend so zu definieren, daß die Ersetzung der $\text{liS}(f)^*)$ in g durch die rechte erst dann vorgenommen wird, wenn, für alle k , $\text{ANB}(k)$ als äquivalenter Unterteilgraph mindestens einmal auftritt. Dies würde gewährleisten, daß jede neue Kante von f min-

*) Wir verwenden diese etwas laxe Sprechweise und meinen aber einen zu $\text{liS}(f)$ äquivalenten Untergraphen

destens einmal als Verbindungskante zwischen dem Restgraph und der eingesetzten rechten Seite auftritt. In der Sprechweise der formalen Graphgrammatiken bedeutet diese Version für *anwend*, daß man z. B. kontextsensitive Produktionen, d. h. $liS(f)$ besteht aus mehr als einer Ecke, auf einfache Weise in kontextfreie umwandeln kann, indem man die Ecken in die Kontextbedingungen, also in den oberen Teil von γ , schiebt.

- 2) Ohne die in 1) erwähnte Alternative zu benutzen, gibt es noch weitere Möglichkeiten, *anwend* anders zu definieren. Hauptsächlich läuft dies auf eine unterschiedliche Auffassung von $liS(f)$ und den Anschlußbedingungen als Untergraph bzw. als Teilgraph hinaus.

2.1) Definition 4/8 verlangt - grob gesprochen - $liS(f)$ als Untergraph und $anb(k)$ als Teilgraph. Eine Möglichkeit wäre, für die Ersetzung der $liS(f)$ zu fordern, daß $liS(f)$ in g nur als Teilgraph vorhanden sein muß. Dadurch ist es z. B. möglich, eine Menge von Manipulationsvorschriften, die gemäß Definition 4/8 auszuführen ist, zu einer einzigen zu vereinen.

2.2) Die Alternative: $liS(f)$ ist Teilgraph und die neuen Kanten werden nur gezogen, wenn die zugehörige Anschlußbedingung als Untergraph in g vorhanden ist, bewirkt nur, daß höchstens weniger verbindende neue Kanten gezogen werden als im Fall 2,1).

Die Fälle aus 2) lassen sich natürlich auch mit der Variation 1) kombinieren, daß also z. B. die Ersetzung der linken Seite *genau dann* vorgenommen wird, wenn $liS(f)$ als Teilgraph *und* die Anschlußbedingung für *jede* neue Kante mindestens einmal auftritt.

- 3) Man kann für die Definition von *anwend* auch z. B. nur fordern, daß eine Ersetzung in g genau dann vorgenommen wird, wenn ein Untergraph u vorliegt, der *homomorphes* Bild von $liS(f)$ ist, d. h. man nimmt Ersetzungen schon

dann vor, wenn nur gewisse Ähnlichkeiten zwischen $liS(f)$ und Teilen von g gegeben sind. Weiterhin könnte man noch für eine sinnvolle Möglichkeit für die Definition von $anwend$ vorsehen, daß für das Ziehen der neuen Kanten k es schon genügt, wenn ein $u'ugg$ existiert, mit $u'hom_{anb}(k)$. Ebenfalls sinnvolle Möglichkeiten ergeben sich, wenn für dieses u' nur $u'tgg$ gefordert wird. Natürlich läßt sich der Fall 3), d. h. daß man Ersetzungen schon dann vornimmt, wenn nur Homomorphismen zwischen $liS(f)$, $anb(k)$ und Teilen von g vorliegen, mit den Fällen 1) und 2) kombinieren.

Schematisch lassen sich die für die Definition der Anwendung eines Graphen (Manipulationsvorschrift) bezüglich eines anderen Graphen sofort einsichtigen Fälle für die Möglichkeiten der Auffassungen für die linke Seite und den Anschlußbedingungen durch die Kombinationen von $\{u_{liS}, tg_{liS}, hom_{liS}\}$ und $\{u_{anb(k)}, tg_{anb(k)}, hom_{anb(k)}\}$ darstellen*), wobei hom_{liS} bedeutet, es gibt einen Homomorphismus von $liS(f)$ auf einen Untergraphen von g . Die anderen Bezeichnungen sind damit selbsterklärend. Der Grund, daß wir in der vorliegenden Arbeit nur den Fall (u, tg) nicht Fall 1) **) , also $liS(f)$ Untergraph und für das Ziehen einer neuen Kante muß die Anschlußbedingung Teilgraph sein, behandeln, ist dadurch motiviert, daß leichter die Parallelen zwischen diesem Ansatz und anderen Graphersetzungsansätzen gezeigt werden können.

*) Nimmt man somit an, daß einmal die Ersetzung *ohne* die in 1) angesprochene zusätzliche Bedingung stattfindet und einmal *mit* dieser, so ergeben sich sofort 18 augenfällige Definitionsmöglichkeiten von $anwend(f, g)$.

**) Diese Kurzschreibweise soll immer eine der möglichen Kombinationen für die Betrachtung der linken Seite und den Anschlußbedingungen bei einer Manipulationsvorschrift beschreiben, zusammen mit einer Information darüber, ob man die Ersetzung vornimmt, wenn nur die Bedingung für die liS erfüllt ist, oder ob auch gleichzeitig die Anschlußbedingung für jede neue Kante mindestens einmal auftreten muß.

Zum Abschluß dieses Kapitels wird noch kurz ein kleines Beispiel behandelt, das einen Eindruck geben soll, welche Konsequenzen die Beschränkung auf die Existenz eines Homomorphismus für das Auffinden der Anschlußbedingung hat.

Beispiel 4/5:

Die Figur 4/14(a) beschreibt eine Formvorschrift f . In der Auffassung $(u, \text{hom}_{\text{anb}(k)})$ nicht Fall 1) wird die neue Kante l gezogen, wenn f auf den Graphen in Figur 4/14(b) angewendet wird, da die (gestrichelt angedeutete) Anschlußbedingung gefunden wird, woraus sich der Graph in Figur 4/14(d) ergibt. Ebenso ist wegen der offensichtlichen Existenz eines Homomorphismus f auf den Graphen der Figur 4/14(c) anwendbar, woraus sich dabei der Graph der Figur 14(e) ergibt.

In unserer gewöhnlichen Auffassung (u, tg) ohne Fall 1) würde f im vorgegebenen Bereich angewendet auf den Graphen in Figur 4/14(b) an diesem nichts verändern. Bei der Anwendung auf Figur 4/14(c) würde sich ebenfalls der Graph der Figur 4/14(e) als Resultat von anwend ergeben. (Die Eckenmarkierungen wurden in den Figuren 4/14(a-e) der Übersichtlichkeit wegen wieder weggelassen).

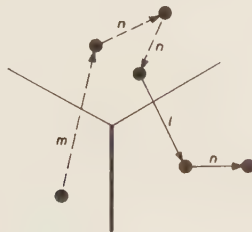


Fig. 4/14(a)

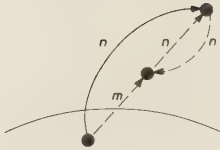


Fig. 4/14(b)

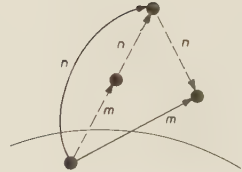


Fig. 4/14(c)



Fig. 4/14(d)

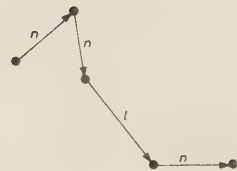


Fig. 4/14(e)



5 EINSTUFIGE UND ZWEISTUFIGE GRAPHMANIPULATIONSSYSTEME

Wir wissen, daß eine Manipulationsvorschrift an sich nur ein Graph ist, den wir auch unter dem Aspekt sehen können, daß er bei geeigneter Aufteilung, bezüglich eines anderen Graphen, als unärer Operator (konkret: als Manipulationsvorschrift) dienen kann. Es ist somit möglich, *Manipulationsvorschriften auch auf Manipulationsvorschriften* anzuwenden. Dies geschieht in völliger Analogie zu Zweistufengrammatiken, bei denen man Ableitungsregeln einer *Metaebene* nach gewissen Vorschriften auf Ableitungsregeln einer *Hyperebene* anwendet. Die Motivation für diese Betrachtung liegt darin, daß man eine *Hyperregel* quasi als 'grobe' Ableitungsvorschrift auffaßt, die erst nach ihrer Veränderung durch *Metaregeln* als brauchbare, *aktualisierte* Regel Verwendung finden kann. So lassen sich auch unendlich viele Regeln auf ein zweistufiges System mit je endlich vielen Regeln reduzieren.

In einem späteren Kapitel wollen wir zeigen, wie man mit Graphen programmiersprachliche Sachverhalte schreiben kann, wobei Graphen sowohl die Strukturen, die wir betrachten wollen, als auch die Operationen auf diesen Strukturen beschreiben sollen. Da z. B. schon die Strukturen in unendlicher Vielfalt auftreten können, muß ein brauchbares System erlauben, mit einem endlichen Satz von Bildungsvorschriften jede gewählte Struktur beschreiben zu können.

Im Prinzip geht es bei der von uns beabsichtigten semantischen Beschreibung nicht um Grammatiken und um von ihnen erzeugte Wortmengen, sondern um die Ersetzungen von Zeichen durch andere im allgemeinen Sinn. Deshalb ist es angebracht, von *Graph-Semi-Thue-Systemen* o. ä. zu sprechen. Im folgenden sollen solche Systeme kurz *Manipulationssysteme* genannt werden und nur bei Bedarf wird näher auf Unterscheidungen über die Form von Manipulationsvorschriften eingegangen.

Definition 5/1:

Ein *einstufiges Manipulationssystem* über den Alphabeten Eckm und Kam ist ein Tripel $eMS = (Eckm, Kam, V)$, wobei V eine endliche Menge von Manipulationsvorschriften ist, denen Graphen über Eckm und Kam zugrunde liegen. Wir nennen die Elemente aus V auch *Produktionen*. ▲

Definition 5/2:

Seien l, g, g' Graphen, $l \xrightarrow{g} g', \lambda \in \Lambda(l, S(f), l)$, wobei f Produktion eines einstufigen Graphmanipulationssystems ist. Genau dann heie g' *mittels f aus g direkt ableitbar*, wenn es ein solches λ und einen Graphen g'' gibt mit $g'' \xrightarrow{\lambda} g'$ und $g'' = \text{anwend}(f, g, \lambda)$ (in Zeichen: $g \xrightarrow{f} g'$). ▲

Definition 5/3:

Sei eMS ein einstufiges Manipulationssystem. Genau dann heie g' *mittels eMS aus g ableitbar*, wenn es eine Folge von Graphen $(g_1, \dots, g_n)$ gibt, mit $g_1 = g$ und $g_n \xrightarrow{\lambda} g'$, und fr jedes i mit $1 \leq i < n$ gibt es ein $f_i \in V$ mit $g_{i-1} \xrightarrow{f_i} g_i$ (in Zeichen: $g \xrightarrow[eMS]{*} g'$). ▲

Wie blich werden wir die Przisierung f bzw. eMS am $\xrightarrow{\quad}$ - Zeichen weglassen, wenn die Bedeutung klar ist. Wird g' erst ber eine Folge von Produktionen Anwendungen aus g abgeleitet, so kennzeichnen wir dies mit $g \xrightarrow{+} g'$.

Definition 5/4:

Es gelte $g \xrightarrow{f} g'$ mit $\text{anwend}(f, g, l) \xrightarrow{\lambda} g'$. Sei $\lambda \in \Lambda(\text{anwend}(f, g, l), g')$, dann heie $\lambda(E_f)$ die *Neuschreibung der Eckenmenge* $N = E(l) \subseteq E(g)$ *mittels f* (in Zeichen: $\text{neu}(N, f)$). ▲

Unter der Neuschreibung der Eckenmenge N versteht man so- mit einfach die Menge der Ecken aus g' , die zu den Ecken der rechten Seite von f korrespondieren, da g' - genauer: Ein zu g' quivalenter Graph g''^{+} - mittels f aus g direkt ableitbar ist. Wir wollen knftig auch die Sprechweise verwenden: N wird mittels f neu geschrieben o. ., wenn

⁺) Wir werden im folgenden weiter diese Unterscheidung fal- len lassen, wenn keine Unklarheiten auftreten knnen.

wir andeuten wollen, daß wir eine Manipulationsvorschrift f auf einen Graphen g in dem Bereich anwenden wollen, der durch N ausgezeichnet ist.

Wenn wir nun Manipulationsvorschriften f selbst auf Manipulationsvorschriften g anwenden wollen, müssen wir uns festlegen, welchem Bereich wir die Neuschreibung zuordnen wollen. Wenn z. B. für die neu zu schreibende Eckenmenge $N: E_1(g)$ gilt, dann könnten wir fordern, daß mit $g \vdash_{\bar{f}} g'$ schließlich $\text{neu}(N, f) \subseteq E_1(g')$ ist. Konkret bedeutet dies: Wenn wir in irgendeinem Bereich des Konnektors $\bar{Y}$ eine Neuschreibung vornehmen wollen, zerlegen wir den neu entstandenen Graphen durch $\bar{Y}$ so, daß alle unverändert gebliebenen Ecken wieder in den Bereich kommen, in dem sie vor der Neuschreibung von f lagen und die neu hinzugekommenen Ecken dem Bereich zugeordnet werden, in dem die neu zu schreibenden Ecken lagen.

Damit werden wir mit einem Problem konfrontiert, das aus der Möglichkeit von Überlappungen resultiert. Betrachten wir z. B. die Manipulationsvorschrift m in Figur 5/1(a) und die Manipulationsvorschrift h in Figur 5/1(b) (Kantenmarkierungen sind der Übersichtlichkeit wegen wieder weggelassen).

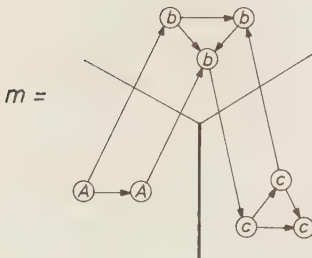


Fig. 5/1(a)

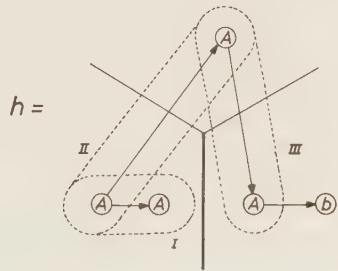


Fig. 5/1(b)

In der Figur 5/1(b) ist durch die gestrichelten Bereiche I, II und III dargestellt, wo Neuschreibungen von h mittels m vorgenommen werden können. Bereich I ist unproblematisch; bei den Bereichen II und III wäre jede Aufteilung der Ecken von $reS(m)$ auf $E_1(h')$, $E_a(h')$ bzw. $E_r(h')$, wenn $h \xrightarrow{m} h'$, reine Willkür.

Somit wollen wir für Manipulationsvorschriften m , die auf andere Manipulationsvorschriften angewendet werden, fordern, daß $|E(Lis(m))|=1$ ist^{*)}. Damit umgehen wir das Problem der Überlappungen und bleiben auch kohärent in der Analogie mit den als kontextfrei vorausgesetzten Metaregeln bei üblichen Zweistufengrammatiken für Zeichenketten.

Mit dieser Einschränkung haben wir die Problematik der Anwendung von Manipulationsvorschriften auf Manipulationsvorschriften noch nicht abgehandelt. Wenn wir wieder die Analogie zu den üblichen Zweistufengrammatiken heranziehen, so wird dort durch die Vorschrift der *"konsistenten Ersetzung"* folgendes gefordert: Wenn man sich für die Anwendung einer bestimmten Metaregel auf eine Hyperregel entschieden hat, muß man diese Metaregel auf *jedes* Vorkommen der linken Seite der Metaregel in der Hyperregel anwenden. Obwohl dieser Vorgang als parallele Anwendung der Metaregel verstanden werden kann, ist er dort unproblematisch, da es keine Einbettungsprobleme gibt. Man kann ihn sequentialisieren, d. h. man kann in beliebiger Reihenfolge ein Auftreten des zu ersetzenden Zeichens durch die rechte Seite der Metaregel ersetzen. Das Ergebnis, also die neugeschriebene Hyperregel, ist stets eindeutig bestimmt.

Im Bereich der Graphmanipulationssysteme ist dies anders. Soll in einer Manipulationsvorschrift h jede Ecke, die

^{*)} Dies ist nicht die einzige, sondern nur die einfachste Alternative.

eine bestimmte Markierung trägt, nach einer gewissen Vorschrift, die durch eine Manipulationsvorschrift m definiert ist, durch einen Graphen ($reS(m)$) ersetzt werden, so kann man zwei Fälle unterscheiden:

- 1) Die neu zu schreibende Ecke tritt nur ein einziges Mal auf: Dieser Fall kann - wie bereits beschrieben - durchgeführt werden.
- 2) Die neu zu schreibende Ecke tritt mehrfach auf:
Hier ergibt sich ein Reihenfolgeproblem dergestalt, daß das Ergebnis der Neuschreibung nach Behandlung jeder neu zu schreibenden Ecke im allgemeinen abhängig von der Reihenfolge ist, in der m auf diese Ecken angewandt wurde.

Beispiel 5/1:

Es folgt ein Beispiel, das zeigen soll, daß das Ergebnis hintereinander ausgeführter Neuschreibungen i. a. abhängig von der Reihenfolge ist. Die Figur 5/2(a) beschreibe eine Manipulationsvorschrift m , die auf eine Manipulationsvorschrift h (Figur 5/2(b)), angewendet werden soll, zunächst bei der Ecke 1 (Ergebnis in Figur 5/3(a)); danach erfolgt die Neuschreibung der Ecke 2 (s. Figur 5/3(b)). Die Figur 5/4(a) beschreibt das Ergebnis von $anwend(f,h)$, wenn zuerst die Ecke 2 von h neu geschrieben wird, danach wird noch die Ecke 1 in Figur 5/4(a) neu geschrieben (Ergebnis in Figur 5/4(b)). Wenn 1 bzw. 2 bereits die Untergraphen von h bezeichnen, die aus der Ecke 1 bzw. 2 bestehen, dann gilt hier $anwend(m,anwend(m,h,1),2) \neq angewend(m,anwend(m,h,2),1)$. Die erhaltenen Graphen sind auch nicht äquivalent.

Die Definition 4/8 verlangt, daß die Eckenbezeichnungsmengen des Graphen der Manipulationsvorschrift und des zu manipulierenden Graphen disjunkt sein müssen. Es versteht sich von selbst, daß wir formal z. B. in $anwend(m,h,1)$ vor der weiteren Anwendung von m die eingesetzte $reS(m)$ umbezeichnen müssen, die zunächst dieselben Bezeichnungen wie

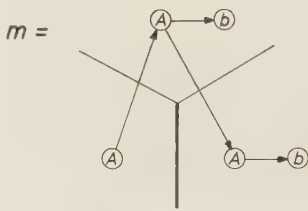


Fig. 5/2 (a)

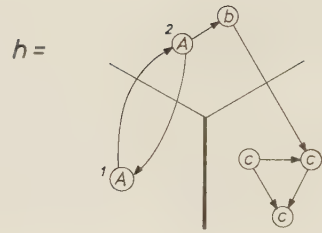


Fig. 5/2(b)

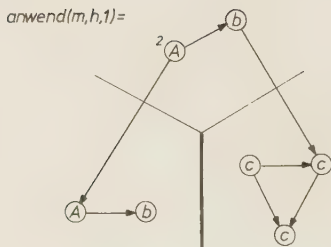


Fig. 5/3(a)

$anwend(m, angewend(m, h, 1), 2) =$

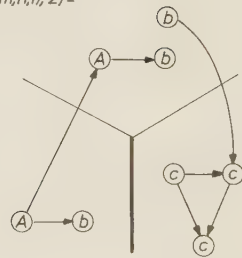


Fig. 5/3(b)

$\text{anwend}(m,h,2) =$

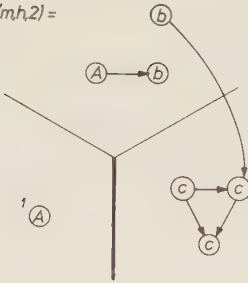


Fig. 5/4 (a)

$\text{anwend}(m, \text{anwend}(m,h,2),1) =$

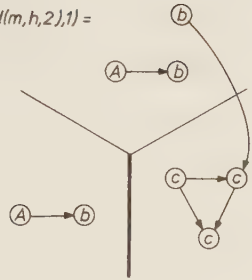


Fig. 5/4 (b)

in m noch trägt.

Die Graphen, die die Figuren 5/3(b) bzw. 5/4(b) darstellen, seien mit $5/3(b)$ bzw. $5/4(b)$ bezeichnet. Man beachte, daß für einen beliebigen Graphen g

$$\text{anwend}(5/3(b), g, 1) \approx \text{anwend}(5/4(b), g, 1)$$

gilt, obwohl $5/3(b)$ und $5/4(b)$ nicht äquivalent sind. Dies ist damit zu erklären, daß die relevanten Anschlußbedingungen für beide Manipulationsvorschriften gleich sind (und linke und rechte Seiten übereinstimmen). ▲

Um dem Problem zu entgehen, zu mehreren Anwendungen, die auf einer Manipulationsvorschrift stattfinden, stets eine Reihenfolge angeben zu müssen, um Eindeutigkeit zu erhalten, und trotzdem ein Analogon zum Begriff der konsistenten Ersetzung zu bekommen, führen wir die Operation der *parallelen Anwendung* $\text{panwend}(m,h)=h'$ einer Manipulationsvorschrift m auf eine Manipulationsvorschrift h ein. Die Definition von panwend besteht aus zwei Teilen. Der erste beschreibt, wie sich bei der Anwendung der h zugrundeliegende Graph verändert, der zweite legt fest, wie die

unverändert gebliebenen und die neu hinzugekommenen Ecken auf die Bereiche der sich ergebenden Manipulationsvorschrift h' zu verteilen sind.

Zur Beschreibung des genannten ersten Teils der Definition von $\text{panwend}(m, h)$ dient die Einführung der Operation $\text{paktual}(m, h)$, die *parallele Aktualisierung* (Anwendung) von m auf h (die sich also nicht um die Eckenauftteilung bzgl. $\vee$ kümmert). Die Operation $\text{paktual}(m, h)$ läßt sich vereinfacht so darstellen, daß $\text{anwend}(m, h, \lambda)$ getrennt für jede Ecke, die dieselbe Markierung wie $\text{lis}(m)$ trägt, auf dem h zugrunde liegenden Graphen durchgeführt wird. Die entstandenen Graphen werden dann zu einem einzigen verschmolzen.

Wir verwenden dazu einen auf [45] basierenden Ansatz, der für Graph-LINDENMAYER-Systeme entwickelt wurde und im folgenden eine Verallgemeinerung erfährt.

Zunächst behandeln wir einen Operator PAKTUAL , der beschreibt, wie eine Menge $\{m_1, m_2, \dots, m_n\}$ von Manipulationsvorschriften 'parallel' auf einen Graphen angewendet werden kann. Es wird zunächst nicht $|E(\text{lis}(m_i))| = 1$ vorausgesetzt. Der Einfachheit halber gelte für alle m_i , daß $\text{lis}(m_i)$ äugh ist. Andere sind für die Beschreibung von PAKTUAL nicht relevant. Man beachte auch, daß die Graphen, die einem m_i und m_j mit $i \neq j$ zugrunde liegen, äquivalent sein dürfen und obendrein für $\text{äq} \in \text{äq}_{\langle m_i, m_j \rangle} \text{äq} (E_t(m_i)) = E_t(m_j)$ für $t \in \{1, r, a\}$ gelten kann, d. h. daß die Eckenauftteilung übereinstimmt und wir somit bis auf die Eckenbezeichnungen dieselbe Manipulationsvorschrift haben können. Wir brauchen dieses 'Kopieren' von Manipulationsvorschriften, um PAKTUAL und paktual einfacher beschreiben zu können. Zusätzlich vereinbaren wir noch, daß die Eckenbezeichnungen der rechten Seite in einer Manipulationsvorschrift m_i mit dem Index der Manipulationsvorschrift, also i , zur Unterscheidung gekennzeichnet sind^{*)}.

^{*)} Diese Kennzeichnung ist o.B.d.A.möglich und dient nur zur bequemerem Darstellung von PAKTUAL .

Man beachte noch, daß für die in Definition 5/5 eingeführten l_i bzw. l_j von g auch $l_i = l_j$ sein darf und für die Graphisomorphismen $\lambda_i \in \text{Äq}(\text{liS}(m_i), l_i)$ bzw. $\lambda_i \in \text{Äq}(\text{liS}(m_i), l)$ durchaus $\lambda_i \neq \lambda_j$ gelten kann. Die Graphen l_i, l_j können sich auch 'überschneiden'.

Definition 5/5:

Es seien $h, l_1, l_2, \dots, l_n$ und die den Manipulationsvorschriften $m_1, m_2, \dots, m_n$ zugrunde liegenden Graphen über Eckm und Kam. $I := \{1, \dots, n\}$ ist eine Indexmenge. Weiterhin gelte, $\forall i \in I$, daß $\text{liS}(m_i) \underline{\text{äq}} l_i, l_i \underline{\text{ugh}}, E(m_i) \cap E(h) = \emptyset$ und $\lambda_i \in \text{Äq}(\text{liS}(f), l_i)$ sei. Die Eckenbezeichnungen von m_i werden zur Unterscheidung mit i gekennzeichnet. Ein Graph h' heißt das Ergebnis der *allgemeinen parallelen Aktualisierung* (in Zeichen: $\text{PAKTUAL}(m_1, \dots, m_n, h, \lambda_1, \dots, \lambda_n)$) g.d.w.

$$h' = \text{con} \left(\left(\bigcup_{i \in I} E(\text{anwend}(m_i, h, \lambda_i)) \right) \setminus \left(\bigcup_{i \in I} E(l_i) \right), \right. \\ \left. \bigcup_{i \in I} K(\text{anwend}(m_i, h, \lambda_i)) \cup \bigcup_{i, j \in I} \text{NEU}(i, j) \right),$$

wobei gilt: Sei $e_i \in E(l_i), e_j \in E(l_j), e_i' \in E_r(m_i), e_j' \in E_r(m_j)$ und $s' \in \text{Kam}$. Dann ist

$$\text{NEU}(i, j) := \{k' \mid (\exists k_i \in K(\text{anwend}(m_i, h, \lambda_i))) (\exists k_j \in K(\text{anwend}(m_j, h, \lambda_j))) \\ (k_i = (e_i', e_j, s') \wedge k_j = (e_i, e_j', s') \wedge k' = (e_i', e_j', s'))\}.$$

Wir vereinbaren noch, daß auch $\text{PAKTUAL}(m_1, \dots, m_n, h, \lambda_1, \dots, l_i, \dots, \lambda_n)$ geschrieben werden darf, wenn $\text{Äq}(\text{liS}(m_i), l_i) \neq 1$ ist, oder nur $\text{PAKTUAL}(m_1, \dots, m_n, h)$, wenn, $\forall i \in I$, genau ein $l_i \underline{\text{ugh}}$ mit $l_i \underline{\text{äq}} \text{liS}(m_i)$ existiert. Zusätzlich vereinbaren wir noch, daß $h' = h$ ist, wenn es ein m_i gibt, für das nicht $\text{liS}(m_i) \underline{\text{äq}} h$ gilt. $\blacktriangle$

Das Konstruktionsverfahren für h' besagt, daß man für PAKTUAL zunächst alle Graphen $\text{anwend}(m_i, h, \lambda_i)$ bildet. Diese werden miteinander verschmolzen im Sinne der üblichen Vereinigung aller Ecken- und Kantenmengen und ergeben einen Graphen, z. B. h'' , aus dem wir die Ecken aller 'alten', zu $\text{liS}(m_i)$ äquivalenten l_i entfernen (" $\bigcup_{i \in I} E(l_i)$ ") (und

damit daranhängenden Kanten $k_i = (e_i', e_j, s')$!). Die 'Anfangs-' bzw. 'Endstücke' dieser Kanten - k_i bildet hier ein Anfangsstück - werden mit 'passenden' End- bzw. Anfangsstücken von Kanten, also z. B. $k_j = (e_i, e_j', s')$, verschmolzen und in $NEU(i, j)$ zusammengefaßt. Die Verschmelzung von passenden 'Halbkanten' und somit der Übergang von h zu h' bei Entfernung von $E(l_i)$ wird in Figur 5/7 dargestellt. Noch ein Hinweis: Trivialerweise ist, $\forall i \in I, NEU(i, i) = \wedge$, wegen der Nichterfüllbarkeit von $k_i = (e_i', e_i, s')$.

Man betrachte zur Erläuterung der Definition 5/5 die Figur 5/5, die grob h beschreiben soll, Die Raute bezeichne den 'konstant bleibenden' Bereich von h , also $restgraph(h, \bigcup_{i \in I} l_i)$. Die Strecken symbolisieren die Menge der Kanten zwischen den Bereichen, l_i bzw. l_j sind die durch m_i bzw. m_j neuzuschreibenden Bereiche.

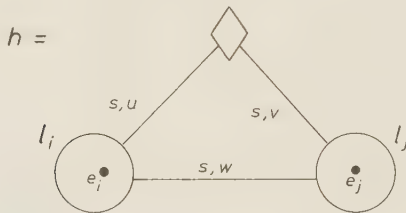


Fig. 5/5

Die Figuren 5/6(a) bzw. 5/6(b) beschreiben ungefähr $anwend(m_i, h, l_i)$ bzw. $anwend(m_j, h, l_j)$. Dadurch 'verschwinden' die gepunktet gezeichneten Bereiche l_i bzw. l_j und es entstehen die Kanten (e_i', e_j, s') bzw. (e_i, e_j', s') .

$anwend(m_i, h, l_j) =$

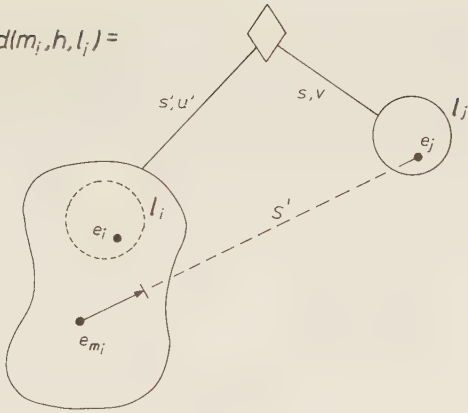


Fig. 5/6(a)

$anwend(m_j, h, l_i) =$

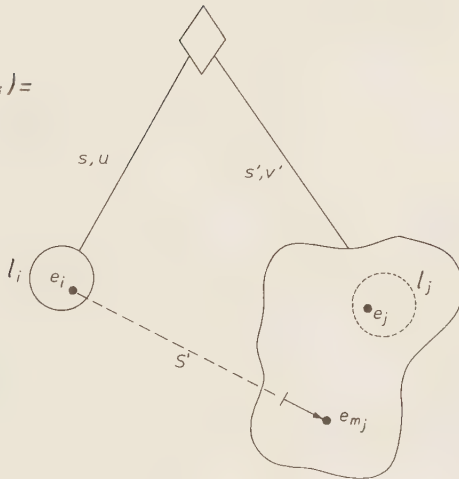


Fig. 5/6(b)

Beide Graphen zusammengenommen liefern die eigentliche Kante (e_i', e_j', s') . anwend(m_i, h, l_i) bestimmte also das 'Anfangsstück' der Kante und anwend(m_j, h, l_j) das 'Endstück'. Dies wird in Figur 5/7 beschrieben.

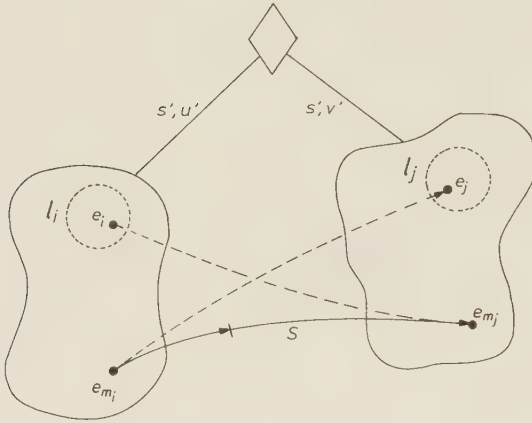


Fig. 5/7

Für das Ziehen einer Kante (e_i', e_j', s') ist also relevant, daß einmal eine Ecke e_j aus einem Bereich l_j Endecke einer Kante (e_i', e_j, s') wird, wobei die Kante ihren Anfangspunkt in e_i' hat, das in $\text{neu}(E(l_i), m_i)$ liegt. Analoges muß für e_j' gelten, d. h. eine Ecke e_i aus einem Bereich l_i muß Anfangsecke einer neuen Kante (e_i, e_j', s') werden, die ihren Endpunkt e_j' in $\text{neu}(E(l_j), m_j)$ hat.

Beispiel 5/3:

Betrachten wir zunächst den Fall, daß zwei Manipulationsvorschriften m_1 (Figur 5/8(a)) und m_2 (Figur 5/8(b)) auf

einen Graphen h (Figur 5/8(c)) mit $\text{PAKTUAL}(m_1, m_2, h, l_1, l_2)$ angewendet werden sollen. (Wir lassen der Übersichtlichkeit wegen wieder Ecken und Kantenbezeichnungen weg.) Dies soll aber an *derselben Ecke* geschehen, also $l_1 = l_2 = l$.

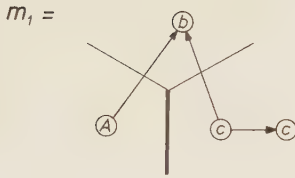


Fig. 5/8(a)

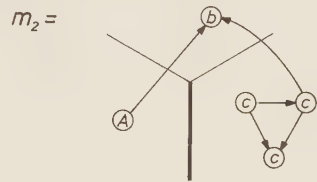


Fig. 5/8(b)

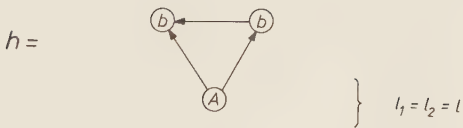


Fig. 5/8(c)

Die Figuren 5/9(a) bzw. 5/9(b) beschreiben $\text{anwend}(m_1, h, l)$
bzw. $\text{anwend}(m_2, h, l)$

$\text{anwend}(m_1, h, l) =$



Fig. 5/9(a)

$\text{anwend}(m_2, h, l) =$



Fig. 5/9(b)

Figur 5/10 beschreibt das Ergebnis $\text{PAKTUAL}(m_1, m_2, h, l, l)$

PAKTUAL(m_1, m_2, h, I, II) =

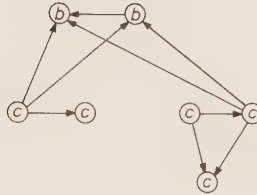


Fig. 5/10

Direkte Kanten zwischen den beiden Neuschreibungen kann es nicht geben, da sich $NEU(1,2)=NEU(2,1)=\emptyset$ ergibt. ▲

Beispiel 5/3:

Wir betrachten nun ein Beispiel, in dem es nicht zu einer vollständigen Identifizierung der zu aktualisierenden Bereiche kommt, sondern nur zu einer 'Überlappung'. Die Figuren 5/11(a) bzw. 5/11(b) beschreiben zwei Manipulationsvorschriften m_1 bzw. m_2 , Figur 5/12 einen Graphen h . Wieder sind Bezeichnungen weggelassen. Durch die gepunktete Umrandung ist angedeutet, wo m_1 bzw. m_2 den Graphen neuschreiben, nämlich in I bzw. II. (Den dritten möglichen Anwendungsbereich betrachten wir nicht*).

Die Ergebnisse von $anwend(m_1, h, I)$ bzw. $anwend(m_2, h, II)$ finden sich in Figur 5/12(a) bzw. 5/12(b). Mit $\hat{=}$ soll angedeutet werden, was den Bezeichnungen aus der Definition 5/5 entspricht. Figur 5/13 präsentiert

PAKTUAL(m_1, m_2, h, I, II).

*) Man beachte, PAKTUAL fordert nicht jede mögliche Aktualisierung, im Gegensatz zum noch zu definierenden paktual.

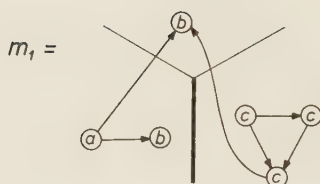


Fig. 5/11(a)

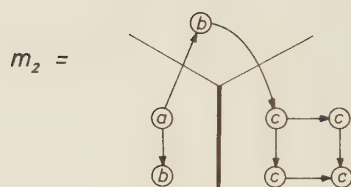


Fig. 5/11(b)

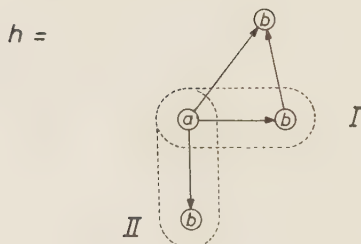


Fig. 5/12

$anwend(m_1, h, I) =$

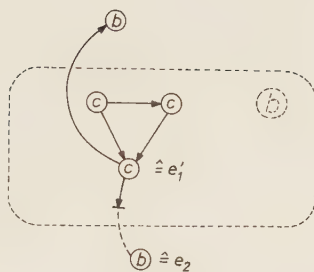


Fig. 5/12(a)

$anwend(m_2, h, II) =$

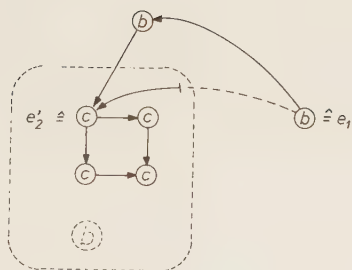


Fig. 5/12(b)

$PAKTUAL(m_1, m_2, h) =$

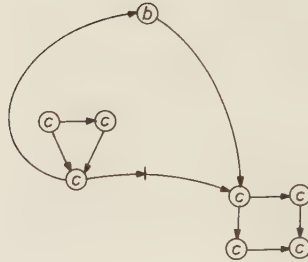


Fig. 5/13

Man beachte, daß sich eine Kante zwischen den beiden Neuschreibungen nur deshalb ergeben hat, weil die neuen Kanten in den Manipulationsvorschriften abwechselnd von der Anschlußecke zur rechten Seite (in m_2) bzw. von der rechten Seite zur Anschlußecke (in m_1) verlaufen. Hätte z. B. m_2 die Gestalt von Figur 5/14(a) besessen, so hätte sich als Anwendung die Figur 5/14(b) ergeben, bei der zusammen mit Figur 5/12(a) die Richtungen der gestrichelten Kanten 'nicht zusammenpassen'. Somit hätte sich als PAKTUAL ein zu Figur 5/13 bis auf die Kante zwischen den beiden Neuschreibungen äquivalenter Graph ergeben.

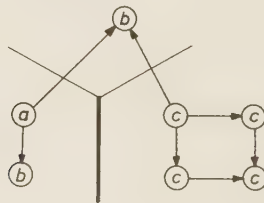


Fig. 5/14(a)

panwend(m,h) geben zu können. Wir bestimmen dabei $h' = \text{paktual}(m,h)$ von dem Graphen, der einer Manipulationsvorschrift h zugrunde liegt und machen noch Aussagen darüber, wie die Ecken von h' bzgl. $\vee$ aufzuteilen sind.

Definition 5/7:

Seien m und h zwei Manipulationsvorschriften und $|E(\text{liS}(m))| = 1$. Unter der *parallelen Anwendung von m auf h* (in Zeichen: $\text{panwend}(m,h)$) verstehe man die Manipulationsvorschrift h', die aus h entsteht, wenn folgende Bedingungen erfüllt sind:

- 1) Der h' zugrunde liegende Graph ist das Ergebnis des in Definition 5/6 beschriebenen $\text{paktual}(m,h)$.
- 2) Sei $t \in \{l, r, a\}$. Dann gilt $(\forall e \in E(h) \cap E(h')) (e \in E_t(h) \Rightarrow e \in E_t(h'))$.
- 3) Sei M wie in Definition 5/6 die Menge der Ecken von h, die wie $\text{liS}(m)$ markiert sind, und wieder $t \in \{l, r, a\}$. Dann gilt $(\forall i \in I) (e_i \in E_t(h) \Rightarrow e_i \in E_t(h'))$, wobei $I = \{1, \dots, n\}$.
- 4) h' ist eine Manipulationsvorschrift. ▲

Die Bedingung 1) ist klar. Bedingung 2) macht eine Aussage darüber, wie die unverändert gebliebenen Ecken von h in h' aufgeteilt werden müssen. In 3) wird vereinbart, wo die Neuschreibungen der Ecken e_i in h' aufzuführen sind, nämlich im gleichen $\vee$ -Bereich, in dem e_i in h lag. Bedingung 4) hat 'Filterwirkung' in zweifacher Hinsicht: Erstens kann $E(\text{reS}(m)) = \emptyset$ sein. Dann könnte bei geeignet gewähltem h und m die Eckenmenge der $\text{liS}(h')$ auch leer sein. Dann ist aber h' keine (sinnvolle) Manipulationsvorschrift im Sinne der Definition 4/1. Zweitens wäre h' aber auch keine Manipulationsvorschrift im Sinne dieser Definition, wenn zwischen $\text{liS}(h')$ und $\text{reS}(h')$ Kanten auftreten würden, was bei geeignet gewähltem h und m ebenfalls der Fall sein kann^{*)}.

^{*)} Damit sind nicht jene Kanten gemeint, die in einem Zwischenschritt (analog zu Figuren 5/15(b-c)) zwischen 'linker' und 'rechter Seite' auftreten könnten.

Betrachten wir ein Beispiel zur Erläuterung der Definition 5/7. Um anzudeuten, daß wir im Unterschied zu Definition 4/8 ("anwend") jetzt eine Manipulationsvorschrift m auf eine *Manipulationsvorschrift* anwenden, also mit panwend , kennzeichnen wir m (wieder in Analogie zu van Wijngaarden-Grammatiken) mit Υ .

Beispiel 5/4:

Figur 5/15(a) repräsentiere eine Manipulationsvorschrift m und Figur 5/15(b) eine Manipulationsvorschrift h . In Figur 5/15(c) ist $\text{anwend}(m,h,1)$ dargestellt, in Figur 5/15(d) $\text{anwend}(m,h,2)$. Die Figur 5/15(d) liefert das Endergebnis $\text{panwend}(m,h)$. Die meisten Eckenbezeichnungen und alle Kantenmarkierungen sind wieder weggelassen.

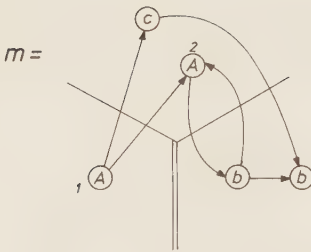


Fig. 5/15(a)

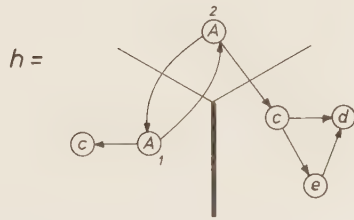


Fig. 5/15(b)

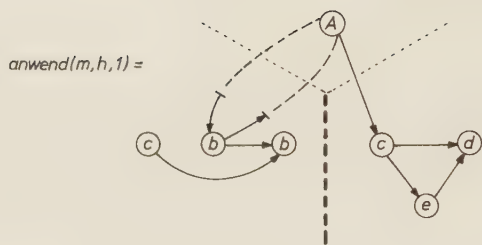


Fig. 5/15(c)

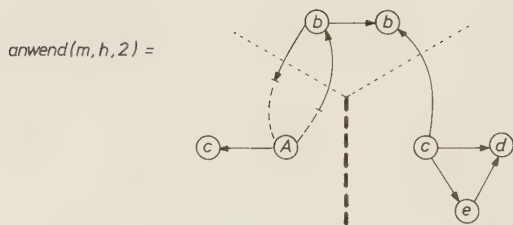


Fig. 5/15(d)

$\text{panwend}(m, h) =$

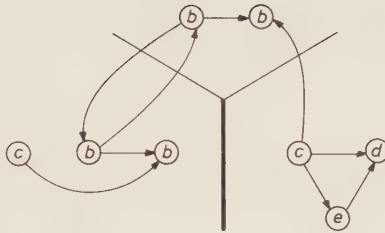


Fig. 5/15(e)

PAKTUAL und paktual liefern für beliebige Manipulationsvorschriften und Graphen g stets ein Ergebnis (evtl. g selbst). panwend muß wegen Bedingung 4) in Definition 5/7 i. a. nicht definiert sein. Um die undefinierten Fälle auszuschließen, definieren wir den Begriff der *Verträglichkeit*, der bei panwend die Bedingung 4) überflüssig machen soll.

Definition 5/8:

Seien m und h Manipulationsvorschriften. m heißt mit h verträglich (in Zeichen: $m\text{verträgh}$), g.d.w. die Gültigkeit der Bedingungen 1), 2) und 3) der Definition 5/7 impliziert, daß $\text{panwend}(m, h)$ eine Manipulationsvorschrift ist. ▲

Die Begriffe der *direkten Ableitbarkeit* und der *Ableitbarkeit* in Definition 5/2 bzw. 5/3 lassen sich auf triviale Weise für panwend übertragen^{*)}. So bedeute künftig $h \xrightarrow{m} h'$, daß der h zugrunde liegende Graph mittels m parallel aktualisiert zu einem Graphen h' mit $h' \text{äq} h$ wurde

^{*)} ebenso Definition 5/1.

und die Eckenaufteilung den Bedingungen 2) und 3) der Definition 5/7 folgt. Der Ableitbarkeitsbegriff ist nur in Verbindung mit Definition 5/8 relevant.

Definition 5/9:

Ein *zweistufiges Manipulationssystem* über den Alphabeten Eckm und Kam ist ein Quadrupel $zMS = (Eckm, Kam, H, M)$, wobei H und M endliche Mengen von Manipulationsvorschriften sind; die Elemente von H heißen *Hypermanipulationsvorschriften* und die von M *Metamanipulationsvorschriften*. Die zugrunde liegenden Graphen der Elemente aus M bzw. H sind Graphen über Eckm und Kam. Wird ein $m \in M$ auf ein $h \in H$ angewendet, so geschieht dies mittels panwend. Weiterhin muß gelten: $(\forall m \in M) (\forall h \in H) (m \text{vertr\"agh} \wedge (h \xrightarrow{+} h' \Rightarrow m \text{vertr\"agh}'))$. ▲

Die letzte Bedingung bedeutet lediglich, daß jedes h' , das aus einem h abgeleitet wurde, wieder mit jedem $m \in M$ verträglich sein muß.

Für die Anwendungen, die wir in dieser Arbeit im Sinne haben, ist es nicht nötig, Angaben über ein Analogon zu machen, das dem Begriff *produktiv* bei gewöhnlichen Zweistufigengrammatiken entspräche. Dort bedeutet es, daß eine Hyperregel erst dann auf eine Zeichenkette anwendbar ist, wenn *alle* nichtterminalen Attribute durch terminale ersetzt sind. Bei uns würde es bedeuten, daß Eckm in zwei disjunkte Mengen $Eckm_N$ und $Eckm_T$ zu zerlegen wäre und Hypermanipulationsvorschriften h erst dann auf einen konkreten Graphen angewendet werden, wenn für all $e \in E(h)$ schließlich $emark(e) \in Eckm_T$ gilt.

Es ist nicht leicht zu entscheiden, ob ein zweistufiges Manipulationssystem 'wohldefiniert' ist in dem Sinne, daß die Metamanipulationsvorschriften mit den Hypermanipulationsvorschriften verträglich sind. Das Problem reduziert sich auf die Frage: Welche Bedingungen müssen die Metamanipulationsvorschriften erfüllen, um mit *jeder beliebigen* Hypermanipulationsvorschrift, auf die sie anwendbar sind, verträglich zu sein? Es lassen sich leicht

Kriterien für m angeben, die notwendig oder hinreichend, aber nicht beides gleichzeitig für beliebiges h sind. Wir treffen nur eine Auswahl und setzen die triviale Bedingung $|E(\text{reS}(m))| \geq 1$ für das folgende voraus.

Satz 5/:

Sei h eine beliebige Manipulationsvorschrift. Wenn die Manipulationsvorschrift m die folgende Bedingung erfüllt, so ist $\text{panwend}(m, h)$ wieder eine Manipulationsvorschrift:
 $(\forall k \in nK(m)) (\exists k' \in aK(m)) (A(k) = ae(k') \vee A(k) = ee(k'))$. ▲

Zwischen einer Anschlußecke und linker und rechter Seite von m muß also eine Kante verlaufen, damit m mit h verträglich ist.

Beweis:

Sei m eine Manipulationsvorschrift, die die Bedingung erfüllt und $h' = \text{paktual}(m, h)$.

Fall 1: $e \in E(h)$ mit $\text{emark}(e) = \text{emark}(E(\text{liS}(m)))$ tritt nur einmal auf.

Fall 1.1: $e \in E_1(h)$.

Die Bilder von $A(k)$ unter den $\varphi_k \in F(k)$ der Definition 4/8 liegen dann höchstens in $E_a(h)$, oder es existieren keine, dann nämlich, wenn $\text{anb}(k)$ in h nicht gefunden wird, also verläuft eine neue Kante höchstens zwischen $E_1(\text{panwend}(m, h))$ und $E_a(\text{panwend}(m, h))^*)$.

Fall 1.2: $e \in E_r(h)$:
analog Fall 1.1.

Fall 1.3: $e \in E_a(h)$:

Die Bilder von $A(k)$ unter den verschiedenen $\varphi_k \in F(k)$ der Definition 4/8 liegen dann höchstens in $E_1(h)$ oder in $E_r(h)$, oder es existieren wieder keine. Somit

*) Für den Fall 1 ist natürlich $\text{panwend}(m, h) = \text{anwend}(m, h, e)$, wenn man nur die zugrunde liegenden Graphen betrachtet.

verläuft auch in diesem Fall keine neue Kante zwischen $E_1(\text{panwend}(m,h))$ und $E_r(\text{panwend}(m,h))$.

Fall 2: $M = \{e \mid e \in E(h) \wedge \text{emark}(e) = \text{emark}.E(\text{liS}(m))\}$ und $|M| > 1$

Fall 2.1: Es gibt keine Kanten zwischen $e_i, e_j \in M$:
Dann kann Fall 1 getrennt auf jedes $e \in M$ angewendet werden.

Fall 2.2: Es gibt Kanten zwischen $e_i, e_j \in M$:

Fall 2.2.1: $e_i, e_j \in E_t(h)$ mit $t \in \{l, r, a\}$:
 $\text{panwend}(m, h) = h'$ bewirkt,
daß $\text{neu}(e_i, m) \cup \text{neu}(e_j, m) \subseteq E_t(h')$
gilt. Also verlaufen die in
 h' auftretenden Kanten zwischen
 $\text{neu}(e_i, m)$ und $\text{neu}(e_j, m)$
zwischen den $e \in E_t(h')$. Die
Kanten zwischen e_i bzw. e_j
und dem 'konstant' bleibenden
Bereich von h ergeben sich
nach Fall 2.1.

Fall 2.2.2: e_i und e_j sind aus verschiedenen Bereichen des Konnektors. Dann darf o.B.d.A nicht
 $e_i \in E(\text{liS}(h))$ und $e_j \in E(\text{reS}(h))$
sein. Deshalb kann in h' auch
wieder keine Kante zwischen
 $\text{liS}(h')$ und $\text{reS}(h')$ auftreten. ▲

Der Satz 5/ soll ein Beispiel dafür sein, daß man beim Entwurf eines zweistufigen Graphmanipulationssystems Meta- und Hypermanipulationsvorschriften auf Verträglichkeit untersuchen muß. Es ist klar, daß bei geeigneter Wahl von h ein m nicht die Bedingung des Satzes 5/ erfüllen muß und trotzdem m verträgh gilt.

6 MANIPULATIONSSYSTEME FÜR QUASIHIERARCHISCHE GRAPHEN

Im Aufbau der vorliegenden Arbeit sind wir nun soweit, daß wir Vorschriften angeben können, die exakt beschreiben, wie ein vorliegender Graph in einen anderen umzuformen ist. Darüber hinaus wissen wir, wie wir solche Manipulationsvorschriften selbst vor ihrer Anwendung auf einen vorliegenden Graphen verändern können. Diese Mechanismen wollen wir für unseren im Kapitel 3 eingeführten Graphentyp untersuchen und die dabei auftretenden Probleme näher betrachten.

Drei Problemtypen ergeben sich bei der Anwendung von Manipulationsvorschriften auf quasihierarchische Graphen: Erstens muß geklärt werden, was wir unter einer 'sinnvollen' Ersetzung in einem quasihierarchischen Graphen verstehen wollen. Welche Kriterien kann man zweitens angeben, die garantieren, daß die Anwendung einer Manipulationsvorschrift auf einen quasihierarchischen Graphen wieder einen quasihierarchischen Graphen ergibt. Drittens wollen wir untersuchen, welche Gestalt eine Metaregel besitzen muß, die bei ihrer Anwendung auf eine Hyperregel die Eigenschaft erzeugt, bzw. nicht zerstört, daß die abgeleitete Hyperregel bei der Anwendung auf einen quasihierarchischen Graphen als Ergebnis wieder einen quasihierarchischen Graphen liefert.

Beginnen wir damit, was wir unter einer 'sinnvollen' Ersetzung bei quasihierarchischen Graphen verstehen wollen.

Ein quasihierarchischer Graph ist ein Graph, der bezüglich der ausgezeichneten Kantenmarkierung in Waldstruktur besitzt. Die Beziehung $e_1 \underline{\text{in}} e_2$ wollen wir aber in der impliziten Darstellung für quasihierarchische Graphen auffassen als Einzeichnen der Ecke e_1 in die Ecke e_2 . Durch die Relation in können verschiedene Sachverhalte dargestellt sein, z. B. im programmiersprachlichen Bereich kann sie bedeuten, daß man vom übergeordneten Modul auf die inneren zugreifen kann. Unter diesem Aspekt ist es oft - rein intuitiv -

wünschenswert, daß mit einer Ecke ihr Inhalt 'verschwindet'*)). Würde man in Figur 3/2 die Ecke 4 ersetzen, so müßte man eine Vereinbarung darüber treffen, ob die Ecken 5 und 6 noch immer in der äquivalenten, impliziten Darstellung in die Ecke 7 gezeichnet werden sollen oder nicht.

Da wir später auch die Manipulation von Graphen mit einer speziellen Kantenmarkierung beschreiben müssen, bzgl. der sie keine Waldstruktur besitzen - im Vorgriff darauf sei bereits hier gesagt, daß es sich dabei um die Hypermanipulationsvorschriften zugrunde liegenden Graphen handelt -, wollen wir zunächst eine Definition angeben, die eine Sonderbehandlung einer speziellen Kantenmarkierung $\underline{m}$ zuläßt: Besitzt der betrachtete Graph bzgl. $\underline{m}$ Waldstruktur, so ergibt sich gerade das gewünschte 'Verschwinden' eines Eckeninhalts.

Definition 6/1:

Sei f eine Manipulationsvorschrift mit einer ausgezeichneten Kantenmarkierung $\underline{m}$. Unter dem *Ergebnis der Anwendung von f auf g in λ bezüglich $\underline{m}$* (in Zeichen: $\text{anwend}(f, g, \lambda, \underline{m})$) verstehen wir den Graphen g' , für den gilt:

Sei $H := \{e | e \in E(g) \wedge (\exists e' \in E(g)) (e' \in \lambda(E_1(f)) \wedge \text{PFAD}(\underline{m}, e, e') \mid \geq 1)\}$.
Es ist $g' = \text{con}(E(\text{anwend}(f, g, \lambda) \setminus H, K(\text{anwend}(f, g, \lambda))))$. ▲

Auf die implizite Darstellung für quasihierarchische Graphen übertragen bedeutet die Modifikation $\text{anwend}(f, g, \lambda, \underline{in})$ anschaulich die Entfernung der Ecken samt Inhalt, ohne daß dieser (als Markierung) explizit angegeben wurde.

Zusammenfassend läßt sich über $\text{anwend}(f, g, \lambda, \underline{m})$ sagen, daß es zunächst wie $\text{anwend}(f, g, \lambda)$ wirkt, dann aber aus g zusätzlich zum zu $\text{lis}(f)$ äquivalenten Untergraphen noch all jene Ecken aus g entfernt werden, von denen ein $\underline{m}$ -Pfad zu einer unter λ auftretenden Bildecke besteht. Anschaulich

*) Wie schon angedeutet, diese Auffassung ist nicht die einzig sinnvolle.

bedeutet dies, daß sich damit die Zahl der Verbindungs-
ecken zwischen dem Restgraphen und der eingesetzten
 $\text{reS}(f)$ reduziert.

(Die hier auftretende Problematik, daß Bilder der An-
schlußbecken von k im eigentlich zu ersetzenden Teil von
 g liegen, sind wir durch unsere Wahl des für jedes k zu
bestimmenden $F(k)$ in Definition 4/8 entgangen: Sei
 $\text{Anker}(k) := \text{con}(\text{anker}(k), \emptyset)$ der Untergraph von g , der nur
aus den Ecken $\text{anker}(k)$ besteht und sei in Definition 4/8
 $F(k) := \{ \varphi_k \mid \varphi_k \in \text{ätg} \langle \text{anb}(k), g \rangle \wedge \text{restrikt} \langle \varphi_k, \text{Anker}(k) \rangle =$
 $\text{restrikt} \langle \lambda, \text{Anker}(k) \rangle \}$,

dann könnte $\varphi_k(A(k)) \in \lambda(E_1(f))$ bei geeignetem f
und λ sein. Diese Variante würde zwar die Möglichkeit
für $\text{anwend}(f, g, \lambda)$ vergrößern, den Umgang damit aber schwie-
riger gestalten.)

Manipulationsvorschriften, die auf der linken Seite bzgl.
einer Relation in 'weniger' spezifiziert sind, 'umfassen'
solche mit stärkerer Spezifikation in dem Sinne, wie ihn
Satz 6/1 präzisiert (s. a. Figur 6/1). Dabei betrachten
wir der Einfachheit halber Manipulationsvorschriften, bei
denen die linke Seite bzgl. in Baumstruktur besitzt.

Satz 6/1:

Sei q ein quasihierarchischer Graph, sowie f und g Manipu-
lationsvorschriften, für die gilt:

1) $\text{liS}(f)$ und $\text{liS}(g)$ besitzen bzgl. in Baumstruktur

2) $f \text{ äug } g$, wobei gilt, $\forall \text{ äug} \in \text{äug} \langle f, g \rangle$:

2.1) $\text{äug}(E_1(f)) \subseteq E_1(g)$

2.2) $\text{äug}(E_x(f)) = E_x(g)$ für $x \in \{a, r\}$.

3) Es gibt ein $\text{lüg}q$ mit $\text{l äq liS}(g)$. (Damit gibt es ein
 $l' \text{ ugl}$ mit $l' \text{ äq liS}(f)$).

4) Sei $\lambda_f \in \text{äq} \langle \text{liS}(f), l' \rangle$ und $\lambda_g \in \text{äq} \langle \text{liS}(g), l \rangle$ mit
 $(\forall x \in E_1(f) \cup \text{liS}(f)) (\lambda_g(\text{äug}(x)) = \lambda_f(x))$.

Dann ist $\text{anwend}(g, q, \lambda_g, \text{in}) \text{ ätg anwend}(f, q, \lambda_f, \text{in})$. $\triangle$

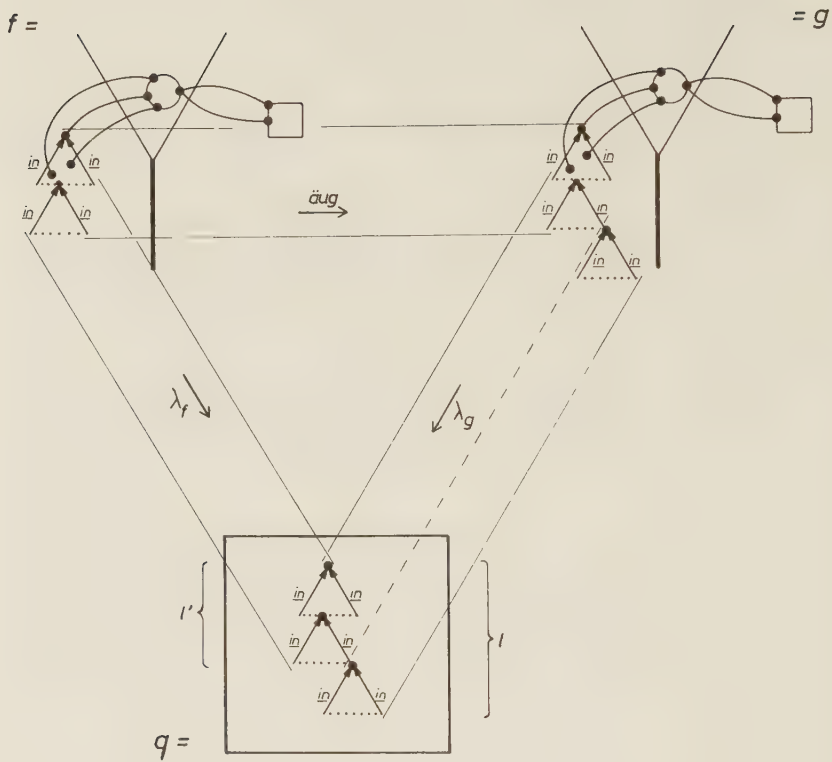


Fig. 6/1

Beweis:

Sei $\ddot{a}ug \in \ddot{a}ug\langle f, g \rangle$ wie in 2). Dann gilt

$$(\forall k \in N(f)) (\exists k' \in N(g)) (ANB(k) \ddot{a}ug ANB(k')).$$

Weiter sei $F_f(k)$ die in Definition 4/8 für $anwend(f, q, \lambda_f)$ zu bestimmende Menge und

$$B_f := \{e \mid e \in E(q) \wedge (\exists k \in N(f)) (\exists \varphi_k \in F_f(k)) (e = \varphi_k(A(k)))\}.$$

Analoges gelte für B_g . Dann ist $B_g \subset B_f$, weil

$$E(q) \setminus \lambda_g(E(lis(g))) \subseteq E(q) \setminus \lambda_f(E(lis(f))).$$

Sei H_f die in Definition 6/1 für $anwend(f, q, \lambda_f, \underline{in})$ zu bestimmende Menge, analog H_g die für $anwend(f, q, \lambda_g, \underline{in})$. Dann ist

$$\text{con}(E(q) \setminus (\lambda_f(E_1(f)) \cup H_f), K(q)) = \text{con}(E(q) \setminus (\lambda_g(E_1(g)) \cup H_g), K(q)).$$

Wegen $B_g \subset B_f$ gibt es in $anwend(g, q, \lambda_g, \underline{in})$ i. a. weniger Kanten zwischen dem Restgraphen und $reS(f)$ bzw. $reS(g)$.

Daraus folgt die Behauptung. ▲

Korollar 6/1:

Unter den in Satz 6/1 gemachten Voraussetzungen ist

$$|E(anwend(f, q, \lambda_f, \underline{in}))| = |E(anwend(g, q, \lambda_g, \underline{in}))|. \quad \blacktriangle$$

Wird in einem quasihierarchischen Graphen q eine Ecke manipuliert, die ein Blatt bzgl. $\underline{in}$ ist, so erhalten wir:

Korollar 6/2:

Ist $E(lis(f)) = \{e\}$ und $emark(e) \neq \varepsilon$, so ist

$$anwend(f, q, \lambda, \underline{in}) = angewend(f, q, \lambda). \quad \blacktriangle$$

Beweis:

$emark(e) \neq \varepsilon$ bedeutet, daß $\text{cont}(e) \in \text{Eckm}$, d. h. es gilt für die in Definition 6/1 zu bildende Menge $H = \emptyset$, da es keine $\underline{in}$ -Kante von einer Ecke zu $\lambda(e)$ gibt. ▲

Die Erweiterung auf Manipulationsvorschriften mit mehr als einer Ecke auf der linken Seite ist einfach:

Korollar 6/3:

Ist $E(\text{liS}(f)) = \{e_1, \dots, e_n\}$ und gilt für alle e_i mit $1 \leq i \leq n$ $\text{emark}(e_i) \neq \varepsilon$, so ist $\text{anwend}(f, q, \lambda, \underline{\text{in}}) = \text{anwend}(f, q, \lambda)$. ▲

Beweis:

Die Argumentation des Beweises von Korollar 6/2 läßt sich auf jede Ecke einzeln übertragen. ▲

Dies mag genügen, um die Problematik der Manipulation von Ecken, die 'inneren Kontext' besitzen, bei quasihierarchischen Graphen zu beschreiben. Gehen wir nun über zum zweiten Problemkreis, der sich mit der Gestalt von Manipulationsvorschriften beschäftigt, die bei ihrer Anwendung auf quasihierarchische Graphen wieder Graphen mit derselben Eigenschaft liefern.

Auch hier, analog zur Diskussion über das Aussehen von Metamanipulationsvorschriften, lassen sich notwendige Kriterien bzw. hinreichende Kriterien angeben, daß bei einer Manipulationsvorschrift f und bei einem quasihierarchischen Graphen q der Ausdruck $\text{anwend}(f, q, \lambda, \underline{\text{in}})$ wieder einen quasihierarchischen Graphen darstellt. Notwendige und hinreichende anzugeben, ist aber nur möglich, wenn man starke Einschränkungen macht.

Das Problem reduziert sich auf die Frage: Welche Manipulationen sind bei Bäumen (oder Wäldern) zulässig, damit das Ergebnis (bzgl. $\underline{\text{in}}$) wieder ein Baum ist (oder ein Wald)? Dazu einige einfache Überlegungen:

Sei f eine Manipulationsvorschrift und q ein quasihierarchischer Graph. Ist $\text{liS}(f)$ kein äquivalenter Untergraph von q , so ist $\text{anwend}(f, q, \lambda, \underline{\text{in}}) = q$. Gilt $\text{liS}(f) \underline{\text{aug}} q$, so muß sicher $\text{reS}(f)$ ein quasihierarchischer Graph sein.

Maximale Ecken der rechten Seite von f dürfen höchstens Anfangsecken einer neuen $\underline{\text{in}}$ -Kante von f sein. Sind sie Endecken von neuen Kanten mit der Markierung $\underline{\text{in}}$, so ist

$\text{anwend}(f, q, \lambda, \underline{\text{in}})$ i. a. kein Baum (siehe dazu Figur 6/2).
Die mit c markierte Ecke müßte schließlich 'in 2 Ecken liegen'.

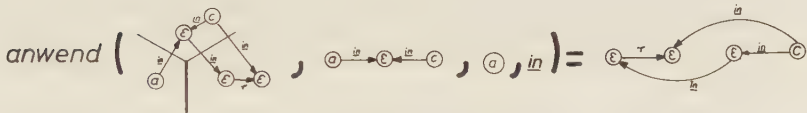


Fig. 6/2

in-Kanten dürfen somit höchstens von den maximalen Ecken aus zu einer Ecke in $E_e(f)$ gehen. Dabei darf höchstens eine in-Kante von jeder maximalen Ecke ausgehen, weil sonst bei geeignetem q von der maximalen Ecke des eingesetzten Graphen zu verschiedenen Ecken des Restgraphen, der nach Entfernung eines zu $\text{lis}(f)$ äquivalenten Untergraphen entsteht, in-Kanten führen würden und damit die Waldstruktur von $\text{anwend}(f, q, \lambda, \underline{\text{in}})$ verletzt wäre.

Dies führt gleich zu der Forderung, daß die Anschlußbedingung zu einer in-Kante im zu manipulierenden Graphen höchstens einmal gefunden werden darf. Die Bedingung 4) des Satzes 6/2 garantiert dies durch eine Aussage über die in-Kanten.

Die Überlegungen führen zu folgender Aussage:

Satz 6/2:

Sei q ein quasihierarchischer Graph und f eine Manipulationsvorschrift. Erfüllt f die folgenden Bedingungen, so ist $\text{anwend}(f, q, \lambda, \underline{\text{in}})$ wieder ein quasihierarchischer Graph.

- 1) $\text{reS}(f)$ ist ein quasihierarchischer Graph,
- 2) $K_{\text{in}}(f) \cap (E_a(f) \times E_r(f)) = \emptyset$,
- 3) Sei Max die Menge der maximalen Ecken bzgl. in in $\text{reS}(f)$. Dann gilt
 $(\forall e \in \text{Max}) (|\{(e, e', \text{in}) \mid e' \in E_a(f)\}| = 1),$
- 4) Sei $F(k)$ wieder die in Definition 4/8 für $k \in K(f)$ eingeführte Menge. Dann gilt
 $(\forall k \in K(f) \cap K_{\text{in}}(f)) (|F(k)| \leq 1). \quad \blacktriangle$

Beweis:

Trivial. $\blacktriangle$

Satz 6/2 führt zusammen mit dem Korollar 6/3 zu folgender Aussage:

Korollar 6/4:

Sind alle Ecken der linken Seite einer Manipulationsvorschrift f Blätter bzgl. in und erfüllt f die Bedingung des Satzes 6/2, so ist $\text{anwend}(f, q, \lambda, \text{in})$ wieder ein quasihierarchischer Graph, wenn q bereits einer war. $\blacktriangle$

Soweit die Ausführungen über die Gestalt von Manipulationsvorschriften, die auf quasihierarchische Graphen angewendet wieder solche als Resultat haben. In unserem dritten Problemkreis wollen wir untersuchen, welche Implikationen sich aus dieser Fragestellung heraus für zweistufige Graphmanipulationssysteme ergeben.

Man kann das Problem so formulieren: Welche Eigenschaften müssen die Hyper- bzw. die Metamanipulationsvorschriften eines zweistufigen Manipulationssystems erfüllen, damit die aktualisierten Hypermanipulationsvorschriften bei der Anwendung auf einen quasihierarchischen Graphen wieder einen solchen als Resultat besitzen? (Daß Hyper- und Metamanipulationsvorschriften miteinander verträglich sein müssen, versteht sich von selbst.)

Es liegt nahe, zwei Positionen zu beziehen: Entweder besitzt die Hypermanipulationsvorschrift h bereits die in

Satz 6/2 geforderten Eigenschaften, oder sie werden erst bei der Aktualisierung erzeugt. Welche Gestalt muß eine verträgliche Metamanipulationsvorschrift m dann besitzen?

Wir beschränken uns auf die Diskussion des Falles, daß h bereits die Eigenschaften von Satz 6/2 erfüllt. Wie muß m dann ausschauen, damit es *keine der Bedingungen* des Satzes 6/2 zerstört?

Richten wir zunächst unser Augenmerk auf das Auffinden einer (sinnvollen) Analogie zum Übergang von $\text{anwend}(m, h, \lambda)$ zu $\text{panwend}(m, h)$, also von $\text{anwend}(m, h, \lambda, \underline{\text{in}})$ zu $\text{panwend}(m, h, \underline{\text{in}})$. Für Spezialfälle haben wir es bereits in den Korollaren 6/2 - 6/4 indirekt getan. Ebenso ist die Analogie trivial für Fälle, bei denen in h nur eine Ecke auftritt, die mit $E_1(m)$ gleich markiert ist.

Wir verzichten auf den leicht darzustellenden Übergang zu einem 'PAKTUAL($m_1, \dots, m_n, q, \lambda_1, \dots, \lambda_n, \underline{\text{in}}$)' (also den möglichen 'mehrfachen' Anwendungen bzgl. $\underline{\text{in}}$) und beschreiben gleich $\text{panwend}(m, h, \underline{\text{in}})$.

Der Graph, der einer Hypermanipulationsvorschrift, die die Eigenschaften von f aus Satz 6/2 besitzt, zugrunde liegt, muß nicht notwendig quasihierarchisch sein. Das ist der Grund, weshalb $\text{anwend}(, , , \underline{\text{in}})$ in Definition 6/1 für den nichtquasihierarchischen Fall definiert sein muß. Somit kann nun beschrieben werden, wie eine Metamanipulationsvorschrift m auf eine *beliebige* Hypermanipulationsvorschrift *bzgl.* $\underline{\text{in}}$ angewendet werden kann.

Definition 6/2:

Seien m und h zwei Manipulationsvorschriften und $|E_1(m)|=1$. Sei $M:=\{e \mid e \in E(h) \wedge \text{emark}(e)=\text{emark}(E_1(m))\}$ und $H:=\{e \mid e \in E(h) \wedge (\exists e' \in M) (|\text{PFAD}(\underline{\text{in}}, e, e')| \geq 1)\}$. Die Manipulationsvorschrift h' heißt das *Ergebnis der parallelen Anwendung von m auf h bzgl. $\underline{\text{in}}$* g.d.w. $h'=\text{con}(E(\text{panwend}(m, h)) \setminus H, K(\text{panwend}(m, h)))$. ▲

Aus $\text{panwend}(m, h)$ werden also noch die Ecken entfernt (und die daranhängenden Kanten), die dieselben Bezeichnungen tragen wie jene in h , von denen aus ein in-Pfad zu einer Ecke aus M geht.

Ist genau eine Ecke in h gleich mit der linken Seite von m markiert, so kann in Definition 6/2 anwend statt panwend geschrieben werden und man erhält somit zusammen mit Definition 6/1 folgende Aussage:

Korollar 6/5:

Seien m und h zwei Manipulationsvorschriften, $E_1(m) = \{e_m\}$ und $\{e_h\} = \{e \mid e \in E(h) \wedge \text{emark}_m(e_m) = \text{emark}_h(e)\}^*$). Dann ist $\text{panwend}(m, h, \underline{\text{in}}) = \text{anwend}(m, h, e_h, \underline{\text{in}})$. ▲

Wie überträgt sich der Verträglichkeitsbegriff von $\text{panwend}(m, h)$ auf $\text{panwend}(m, h, \underline{\text{in}})$?

Satz 6/3:

Seien m und h zwei Manipulationsvorschriften. Ist m mit h unter $\text{panwend}(m, h)$ verträglich, dann ist auch m mit h unter $\text{panwend}(m, h, \underline{\text{in}})$ verträglich. ▲

Beweis:

Da m mit h verträglich ist, gibt es keine Kanten zwischen $\text{liS}(\text{panwend}(m, h))$ und $\text{reS}(\text{panwend}(m, h))$. Durch Wegnahme von Ecken können keine erzeugt werden. ▲

Die Umkehrung des Satzes ist nicht richtig; m kann mit h unter $\text{panwend}(m, h, \underline{\text{in}})$ verträglich sein, aber unter $\text{panwend}(m, h)$ nicht. Dazu betrachten wir ein Beispiel mittels der Figuren 6/2(a-d). (Man beachte auch, daß m nicht die Bedingung des Satzes 5/ erfüllt!)

*) d. h. es gibt genau eine Ecke in h , die wie die linke Seite von m markiert ist.

$h =$

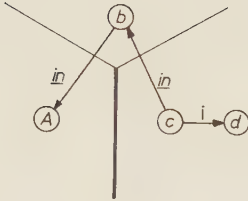


Fig. 6/7(a)

$m =$

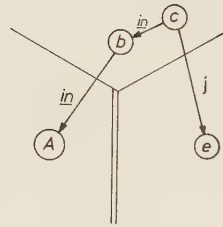


Fig. 6/7(b)

$panwend(m, h) =$
 $(= anwend(m, h) =)$

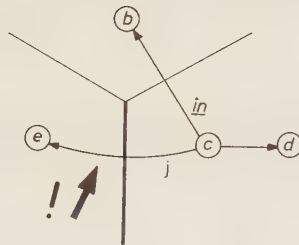


Fig. 6/2(c)

$\text{panwend}(m, h, \underline{\text{in}}) =$
 $(= \text{anwend}(m, h, \underline{\text{in}}) =)$

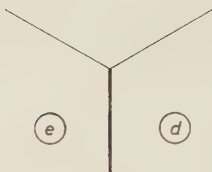


Fig. 6/2 (d)

Die Übertragung des Ableitbarkeitsbegriffs (Definition 5/2) und der Verträglichkeit (Definition 5/8) auf $\text{anwend}(m, h, \lambda, \underline{\text{in}})$ bzw. $\text{panwend}(m, h, \underline{\text{in}})$ nehmen wir als gegeben an.

Definition 6/3:

Sei f eine Manipulationsvorschrift. Wir nennen f *erhaltend*, g.d.w. für alle quasihierarchischen Graphen q gilt, daß $\text{anwend}(f, q, \lambda, \underline{\text{in}})$ wieder ein quasihierarchischer Graph ist. $\blacktriangle$

Definition 6/4:

Ein *zweistufiges Manipulationssystem* für quasihierarchische Graphen über den Alphabeten Eckm und $\text{Kam} \cup \{\underline{\text{in}}\}$ ist ein Quintupel $\text{zMSq} = (\text{Eckm}, \text{Kam}, \underline{\text{in}}, H, M)$, für das gilt:

- 1) $(\text{Eckm}, \text{Kam}, H, M)$ ist ein zweistufiges Manipulationssystem, bei dem $\text{panwend}(m, h)$ durch $\text{panwend}(m, h, \underline{\text{in}})$ ersetzt wird und
- 2) jedes h' , das aus $h \in H$ ableitbar ist, *erhaltend* ist. $\blacktriangle$

Es wird noch ein Kriterium für Metamanipulationsvorschriften angegeben, die bei ihrer parallelen Anwendung auf eine Hypermanipulationsvorschrift, die die Eigenschaften des Satzes 6/2 erfüllt, diese Eigenschaften nicht zerstören.

Satz 6/4:

Sei q ein quasihierarchischer Graph, m und h seien Manipulationsvorschriften, für die gilt:

- 1.1) h erfüllt Satz 6/2,
- 1.2) $(\forall k \in K(h)) (|\text{PFAD}(\underline{\text{in}}, e, A(k))| \leq 1 \wedge e \in E_1(h))^*$,
- 2.1) m erfüllt Satz 6/2,
- 2.2) $\{e_1\} = E_1(m) \wedge \text{emark}_m(e_1) \neq \varepsilon$,
- 2.3) m erfüllt Satz 5/.
- 3) $(\forall k \in K(h) \cap K_{\underline{\text{in}}}(h)) (e = AE(p) \wedge e \in E_1(h) \wedge p \in \text{PFAD}(\underline{\text{in}}, e, A(k)) \Rightarrow \text{emark}_h(e) \neq \text{emark}_m(e_1))$

Es gilt: $\text{anwend}(\text{panwend}(m, h, \underline{\text{in}}), q, \lambda, \underline{\text{in}})$ ist ein quasihierarchischer Graph. Δ

Es sind noch Variationen zu den Voraussetzungen für h und m denkbar. Aufgrund der rigiden Bedingung, daß der aus q abgeleitete Graph (bzgl. $\underline{\text{in}}$) Waldstruktur besitzen muß, und der relativ schwachen Anforderungen an h , ist es fast selbstverständlich, daß für m sehr starke Einschränkungen gemacht werden müssen, damit die Aussage von Satz 6/4 möglich wird.

Beweis:

Es genügt zu zeigen, daß m nicht die Bedingung 1.1) zerstört. Dies wird durch 1.2) - 3) garantiert.

Wegen 2.3) und Satz 6/3 ist $\text{panwend}(m, h, \underline{\text{in}})$ sicher wieder eine Manipulationsvorschrift.

2.2) besagt, daß m nur auf Blätter angewendet werden kann (und somit, wegen 1.2), nicht auf Anschlußecken). Ist $e_1, e_2 \in \text{Blatt}_h(\underline{\text{in}})$ und $\text{emark}(e_1) = \text{emark}(e_2) = \text{emark}(e_1)$, so gibt es keine $\underline{\text{in}}$ -Kanten zwischen der für e_1 bzw. e_2 eingesetzten rechten Seite von m wegen 2.3) und 2.1). Die $\underline{\text{in}}$ -Kanten zwischen einer für e_1 eingesetzten rechten Seite

*) $\text{emark}_h(ee(k)) = \varepsilon$

von m und dem 'konstanten' Teil von h ergibt sich aus $\text{anwend}(m, h, e_1, \underline{\text{in}})$ (s. a. Definition 6/2 und 5/5). Da m nach 2.1) den Satz 6/2 erfüllt, gilt für $p := \text{panwend}(m, h, \underline{\text{in}})$ die Aussage, daß $\text{reS}(p)$ ein quasihierarchischer Graph ist. Somit ist für p die Bedingung 1) des Satzes 6/2 erfüllt.

Wegen 2.1) kann auch keine in-Kante von $E_a(p)$ nach $E_r(p)$ verlaufen (somit ist 2) von Satz 6/2 erfüllt).

m erfüllt wegen 2.1) insbesondere die Bedingung 4) des Satzes 6/2. Also gibt es von einem $e \in E_r(p)$ höchstens eine in-Kante zu einem $e' \in E_a(p)$ (also gilt 3) von Satz 6/2).

Die Bedingung 1.2), also die Existenz eines 'in-Such-Pfades' ist eine schärfere Formulierung der Bedingung 4) des Satzes 6/2. Da m wegen 3) weder auf Ecken, die am Anfang eines solchen Pfades liegen, angewendet werden kann, noch auf Ecken mit ϵ -Markierung, bleibt die Bedingung 4) von Satz 6/2 erhalten.

Somit ist $\text{panwend}(m, h, \underline{\text{in}})$ eine Manipulationsvorschrift, die sämtliche Anforderungen des Satzes 6/2 erfüllt und deshalb $\text{anwend}(\text{panwend}(m, h, \underline{\text{in}}), q, \lambda, \underline{\text{in}})$ ein quasihierarchischer Graph. ▲

Wir können nun auf das o.g. p noch weiter Metamanipulationsvorschriften anwenden und erhalten:

Korollar 6/6:

Sei h und $\{m_1, \dots, m_n\}$ eine Menge von Manipulationsvorschriften, die die Bedingungen des Satzes 6/4 erfüllen, sowie q ein quasihierarchischer Graph. Dann ist

$\text{anwend}(\text{panwend}(m_1, \text{panwend}(m_2, (\dots, \text{panwend}(m_{n-1}, \text{panwend}(m_n, h, \underline{\text{in}}), \underline{\text{in}}) \dots), \underline{\text{in}}), \underline{\text{in}}), q, \lambda, \underline{\text{in}})$
ein quasihierarchischer Graph. ▲

Die Theoreme in diesem Kapitel machen deutlich, daß es schwierig ist, allgemein gültige Kriterien beim Umgang mit zweistufigen Manipulationssystemen anzugeben. Es muß aber darauf

hingewiesen werden, daß ein betrachtetes System bezüglich der Meta- und Hypermanipulationsvorschriften nicht notwendigerweise die angegebenen Einschränkungen besitzen muß und es trotzdem die Eigenschaft 'quasihierarchisch' erhält.

In diesem Sinne haben die angegebenen Sätze und Korollare theoretischen Charakter und stellen nur ein Hilfsmittel zum Aufzeigen der Probleme dar.

Ein wichtiger Aspekt kam bei der Behandlung von formalen Manipulationssystemen für Graphen mit einer Spezialmarkierung zum Vorschein: Die Rücksichtnahme auf die unterliegende Bedeutung dieser Kantenmarkierung führt schnell zu formalen Problemen, wenn man die Wohldefiniertheit der eingeführten Begriffe für Spezialfälle sichern will, z. B. $\text{anwend}(f, g, \lambda, \underline{\text{in}}) \stackrel{!}{=} \text{anwend}(f, g, \lambda)$ wenn Blätter bzgl. in zu aktualisieren sind. Dies ist nur mit einem erhöhten Aufwand, z. B. gemessen an der Zahl der zur Beschreibung notwendigen Zeichen o. ä., möglich. Dafür umgeht man später einen Aufwand zur Beschreibung eines Sachverhalts, bei dem z. B. eine spezielle Relation in von Bedeutung ist. Auf dieses Phänomen der Verlagerung des Aufwandes der semantischen Beschreibung gehen wir im folgenden Kapitel ein.

7 PROBLEME DER SEMANTISCHEN BESCHREIBUNG

In den vorangegangenen Kapiteln beschäftigten wir uns mit Zeichenketten, Graphen genannt, denen wir zum Teil noch das Attribut quasihierarchisch gegeben hatten. Wir behandelten einmal ihre *äußere Gestalt* und beschäftigten uns auch mit der *Bedeutung* der Zeichenkette $\text{anwend}(f,g,\lambda)$. Dazu fand man noch an einigen Stellen Äußerungen über die *Brauchbarkeit* dieser Konstrukte.

Diese drei Aspekte tauchen auch im Bereich der Linguistik und bei den Programmiersprachen auf, genannt *Syntax*, *Semantik* und *Pragmatik*. Die Unterscheidung geht auf den amerikanischen Philosophen Charles MORRIS zurück^{*)}. Er präzisiert: Syntax behandelt die Kombinierbarkeit der Zeichen ohne Rücksicht auf ihre spezielle Bedeutung oder ihre Beziehung zur Umwelt. Semantik behandelt die Bedeutung der Zeichen in allen Arten des Bezeichnens, Pragmatik behandelt die Herkunft, den Gebrauch und die Wirkung der Zeichen innerhalb der Umgebung, in der sie auftreten.

Den Aspekt der Pragmatik wollen wir im folgenden nicht weiter diskutieren, weil er schließlich nur mit vagen Begriffen - nicht zuletzt unter dem Gesichtspunkt der Ästhetik - abzuhandeln ist.

Mit den Begriffen Syntax und Semantik von Zeichen gerät man bereits bei der Unterscheidung dessen, was noch in den Bereich des einen oder des anderen gehört, in Schwierigkeiten. Bei der Durchsicht der Literatur und der Veränderung in den Auffassungen über die Begriffe läßt sich leicht die Neigung feststellen, daß man diejenigen Aspekte von Zeichen(ketten) gerne mit Syntax bezeichnet, die sich formal (im Sinne der Existenz eines den Aufbau der Zeichenketten beschreibenden Systems von Regeln) darstellen lassen. Die Aspekte, bei denen dies nicht möglich ist, werden im Teil Semantik behan-

^{*)} nach [62]

delt.

Ein Beispiel dazu ist die Definition der Programmiersprache ALGOL 60, [46]. Dort werden die Eigenschaften, die korrekte ALGOL 60-Programme besitzen müssen und die mittels kontextfreier Grammatiken beschreibbar sind, unter der Überschrift 'Syntax' erklärt, die 'sonstigen' Eigenschaften in 'Semantik'.

Durch die Entwicklung mächtigerer Kalküle, z. B. der *Zweistufengrammatiken*, [73], wurde es möglich, offensichtlich zur Syntax gehörige Dinge, wie: "Vor dem angewendeten Auftreten eines Identifikators muß ein definierendes Auftreten stehen", tatsächlich im Teil Syntax abzuhandeln. Eine scharfe Trennung der Bereiche ist jedoch noch nicht gemacht worden und auch schwierig, denn z. B. im linguistischen Bereich ist der Plural ein sprachlicher Aspekt, der nicht nur syntaktischer, sondern auch semantischer Natur ist. In [37]^{*)} wird einfach dargestellt, wie man eine 'syntaxlose' Programmiersprache definieren kann und damit gezeigt, daß eine Abtrennung zwischen Syntax und Semantik willkürlich bleibt.

Nachdem es letztlich nicht gelungen ist, zu definieren, was Syntax und Semantik einer Sprache, also einer (i. a. unendlichen) Menge von Zeichenketten *ist*, stellt man Bedingungen auf, welche Eigenschaften Syntax und Semantik *besitzen müssen*. Man fordert, die Syntax muß *entscheidbar* und die Semantik *berechenbar* sein. Man meint mit dem ersten, daß es einen Algorithmus geben muß, der in endlicher Zeit entscheidet - wenn er die Beschreibung der Gestalt der Zeichenketten einer Sprache und eine konkrete Zeichenkette erhält - ob diese Zeichenkette in dieser Sprache enthalten ist oder nicht.

Mit der zweiten Forderung, der Berechenbarkeit, gelangt der Begriff der Semantik in einen Bereich, in dem er 'letztlich'

*)

nicht mehr definierbar ist. Dabei muß man sich vor Augen halten, daß alle Definitionen von Berechenbarkeit (TURINGmaschinen, rekursive Funktionen, POSTsche Systeme, MARKOValgorithmen u.s.w.) nichts weiter als Versuche sind, den *intuitiven* Begriff der Berechenbarkeit zu präzisieren, der 'an sich' nicht definierbar ist. Daran ändert auch die Autorität von CHURCH nichts ("CHURCHsche These"). In diesem Sinne entzieht sich schließlich die Bedeutung eines Zeichens jedem Beschreibungsversuch oder wie es BÖHM ausdrückt: "Der Sinn eines Zeichens liegt außerhalb der Zeichen, transzendiert sie"*) .

Als es GÖDEL gelang, in seinen Arbeiten über formal unentscheidbare Sätze den Glauben - wie ihn HILBERT vertrat - zu erschüttern, daß für jedes mathematische Problem entweder die Existenz oder die Nichtexistenz einer Lösung angegeben werden kann, besann man sich darauf, die Bedeutung von Zeichen letztlich auf 'verlässliche' elementare Grundoperationen zurückzuführen, z. B. auf den Prozeß des Zählens in der rekursiven Analysis oder durch Schreiben und Löschen von Zeichen bei TURINGmaschinen**) (oder auf die anderen Berechenbarkeitsbegriffe).

Halten wir fest: Semantik betreiben heißt, zu einer Zeichenkette angeben, welche Folge von (ausführbaren) Grundoperationen ihr zugeordnet wird. Diese Maxime verhindert nur 'Trivialsemantiken' etwa dergestalt, daß als Bedeutung einer Zeichenkette xyzuv... die Zeichenkette x'y'z'u'v'... (und sonst nichts) angegeben wird, sie bedeutet aber nicht, daß sich jede semantische Beschreibung *unmittelbar* auf die Grundoperationen beziehen muß.

Irgendwo in dem weiten Bereich zwischen der Zeichenmenge selbst und den rekursiven Funktionen nebst deren Äquivalenten sind alle Kalküle für die Beschreibung der Semantik von

*) aus [39]

**) s.a. [29]

Elementen einer Zeichenmenge angesiedelt. Dort darf jeder Semantiker behaupten, daß die Zeichenketten seines Kalküls *die* Semantik eines gegebenen Sachverhalts darstellen.

Da der semantische Kalkül auch durch eine Menge von Zeichenketten dargestellt wird, ist es angebracht, wiederum von Syntax, Semantik und Pragmatik - diesmal vom semantischen Kalkül - zu sprechen. In diesem Zusammenhang wird die immer wieder schmerzlich empfundene Tatsache erklärlich, die eine Art *perpetuum mobile* der Semantik darstellt: Hat man ein relativ ausdrucksstarkes Beschreibungsmittel, so ist der Aufwand (z. B. gemessen an der Zahl der notwendigen alphanumerischen Zeichen) für die Beschreibung der Semantik des Mittels hoch, dafür die Beschreibung eines gegebenen Sachverhalts i. a. niedrig; ist das Beschreibungsmittel nicht ausdrucksstark, so bedarf es wenig Aufwands zur Beschreibung der Semantik des Mittels, aber viel Aufwand für die Beschreibung des gegebenen Sachverhalts.

Es ist klar, daß es sinnvoll ist, sich ein Beschreibungsmittel so zu definieren, daß man n Sachverhalte einfach beschreiben kann und nur einmal die Beschreibung des Beschreibungsmittels angeben muß. Daraus legitimiert sich die Erstellung eines semantischen Kalküls.

Ein Versuch der Rechtfertigung für den in dieser Arbeit beschriebenen Kalkül, der Semantik durch Veränderungen in Graphen beschreiben will, wurde in der Einleitung gegeben. Weitere pragmatische Kriterien beim Entwurf eines Semantikalküls für Programmiersprachen sind z. B. auch: Unterstützt der Kalkül den Programmentwurf, die Programmverifikation u.s.w.. Im Rahmen dieser Arbeit sollen aber solche Aspekte nicht weiter untersucht werden.

8 SEMANTIKBESCHREIBUNG DURCH MANIPULATIONSSYSTEME FÜR QUASIHIERARCHISCHE GRAPHEN; BEISPIELE

In diesem Kapitel wollen wir anhand einiger ausgewählter Beispiele zeigen, wie die Semantik programmiersprachlicher Sachverhalte durch Manipulationssysteme für quasihierarchische Graphen dargestellt werden kann.

In der Literatur gibt es verschiedene weitere Ansätze zur Semantikbeschreibung, die in Kapitel 10.2 diskutiert werden. Gemeinsame Aspekte, die auch für das Verständnis des folgenden wichtig sind, behandeln wir hier.

Eine mögliche Klassifikation ist die Einteilung der Darstellungen in *interpreterorientierte* und *compilerorientierte* Beschreibungen (s. z. B. [4]).

Die *interpreterorientierte Semantik* erklärt die Bedeutung eines Programms durch seine Wirkung auf die externen Eingabedaten, die zusammen mit der 'Interpretation' eines vorgegebenen Inkrementes von Programmtext transformiert werden. Dem Interpretationsvorgang wird eine (abstrakte) Maschine zugrunde gelegt, deren Operationen als unmittelbar verständlich angesehen werden.

Bei der *compilerorientierten Semantik* wird ebenfalls das Programm analysiert und in seiner Gesamtheit in eine Form gebracht, von deren 'Einzelaktionen' man annimmt, daß sie einfacher und verständlicher sind^{*)}. Im Gegensatz zur interpreterorientierten Semantik werden aber erst *nach* der vollständigen Übersetzung dem Programm in der übersetzten Form Daten angeboten.

Die Gesamtheit, Übersetzung des Ausgangsprogramms und Lauf

^{*)} Damit soll nicht behauptet werden, daß die von einem Compiler erstellte Version eines Programms in seiner Gesamtheit 'verständlicher' ist als die Fassung in einer höheren Programmiersprache.

der übersetzten Form, bildet auch bei der compilerorientierten Semantik einen Interpreter. Verlagert man die Veränderung der Daten bei der interpreterorientierten Semantikbeschreibung an das 'Ende' des Interpretiervorgangs, so wirkt der Gesamtinterpretierer in der Hauptsache als Compiler. In diesem Sinne verwischen sich wieder die Unterschiede zwischen den beiden Semantikbeschreibungskategorien.

Beide Darstellungsarten wollen von der äußeren Form des Programms ausgehend dessen Bedeutung beschreiben. Dabei wird das Prinzip deutlich, einer syntaktischen Produktion eine^{*)} oder eine Kette von semantischen Produktionen (Manipulationen) zuzuordnen. Bekannte Beispiele für diese Vorgehensweise sind der ALGOL 68-Report [73], bei dem sich einem Abschnitt mit syntaktischen Produktionen gleich ein Abschnitt Semantik anschließt, oder in der Beschreibung von EULER in [75], bei der zu den syntaktischen Produktionen semantische Funktionen angegeben sind.

In [26] wird der Vorschlag gemacht, daß man, da Abgrenzungen zwischen Syntax und Semantik stets willkürlichen Charakter haben^{**)}, ein einheitliches Beschreibungsmittel für beide Sprachaspekte wählen soll, um das Problem der Abgrenzung zu beseitigen. Diese Absicht findet aber durch das oben Gesagte eine weitergehende Bestätigung: Je ähnlicher das Vorgehen bei der Syntax- und Semantikbeschreibung ist, um so einfacher ist es, das genannte Prinzip zu verwirklichen, durch Zuordnung von semantischen Manipulationen zu syntaktischen Produktionen Semantik zu beschreiben.

Im folgenden gehen wir davon aus, daß wir einen Semantikbeschreibungskalkül entwerfen wollen, der die Semantik eines Programms durch einen quasihierarchischen Graphen angeben

^{*)} Es genügt, die Zuordnung einer einzigen Semantikproduktion zu einer syntaktischen Produktion zu fordern, denn es ist stets möglich, dies durch Einführung von entsprechenden ad hoc-Produktionen zu erzwingen.

^{**)} s. a. Kap. 7.

will. Wurde von der Syntaxanalyse die Anwendung einer Produktionsfolge $p_1 p_2 \dots p_n$ (z. B. in der kanonischen Form) zur Erstellung des Programms erkannt, so werden auf einen 'Startgraphen' die zugeordneten Semantikmanipulationen angewendet. Der dabei entstehende Graph repräsentiert die Semantik des übersetzten Programms. Mit dieser bei Semantikbeschreibungen üblichen Vorgehensweise stellen wir uns auf den pragmatischen Standpunkt, daß ein Interpreter für die 'Abarbeitung' des Semantikgraphen sehr einfach in seiner Funktionsweise ist. Er muß nur Muster erkennen und Eckeninhalte (real: Speicherinhalte) transferieren^{*)}).

Wenn keine Mehrdeutigkeiten auftreten können, wollen wir zur Darstellung der Manipulationsvorschriften, die die Veränderungen des Semantikgraphen beschreiben, einige Vereinfachungen verwenden.

Es wird eine Mischung aus der expliziten und impliziten Darstellung für quasihierarchische Graphen verwendet.

Ist eine Ecke einer Manipulationsvorschrift h unmarkiert, so soll dies bedeuten, daß sie *bei der Anwendung von h* , für eine Ecke mit beliebiger Markierung steht. Analoges gelte für die Kanten. Man erspart sich dadurch die mehrfache Angabe einer Anschlußbedingung^{**)}).

Enthält eine Manipulationsvorschrift einen Anteil, der äquivalent zu Figur 8/1(a) ist, so soll dies zusammen mit den bereits bestehenden Vereinbarungen eine Abkürzung für Figur 8/1(b) sein und bedeuten: Die von der zur äquivalenten, angegebenen Ecke der linken Seite zu einer Ecke e des

^{*)} In unserer Sprechweise können wir diesen Sachverhalt so darstellen: Wir haben quasihierarchische Graphen definiert (Definition 3/) sowie eine Repräsentierung (explizit/implizit) angegeben und verzichteten auf die Beschreibung einer Interpretation.

^{**)} s. a. [43]. Ist $Eck_m = \{m_1, m_2, \dots, m_n\}$, so entspräche die Nichtmarkierung einer Ecke hier der Angabe von $m_1 \vee m_2 \vee \dots \vee m_n$ als Eckenmarkierung dort.

'konstanten' Teils auslaufenden Kanten mit der Markierung m sollen nun von der angegebenen Ecke der eingesetzten Ecke auslaufen. Analoges gelte für die einlaufenden Kanten. Ist $|Eckm|=i$ und $|Kam|=j$, so repräsentiert Figur 8/1(a) die (bei der vorgegebenen Darstellung) für die exakte Beschreibung maximal $2ij$ notwendigen Anschlußbedingungen. (Es lassen sich noch leicht weitere vereinfachte Repräsentierungen von Manipulationsvorschriften angeben, für unsere Zwecke im folgenden sind sie aber unnötig.)

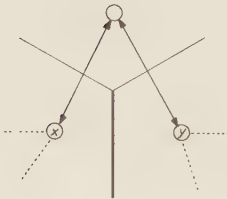


Fig. 8/1(a)

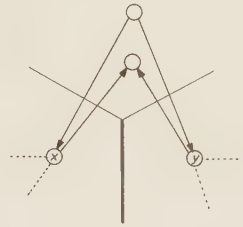


Fig. 8/1(b)

Bestimmte Ecken- bzw. Kantenmarkierungen werden durch die Form der Ecken- bzw. Kanten bereits dargestellt.

Man beachte, daß die dargestellten Manipulationsvorschriften i. a. die Bedingungen der entsprechenden Sätze aus den Kapiteln 5 und 6 erfüllen, so daß keine nichtquasihierarchischen Graphen als Zwischenschritte und als Ergebnis auftreten können. Bei den Manipulationsvorschriften, die die Bedingungen nicht erfüllen, ist der Nachweis einfach, daß ebenfalls keine nichtquasihierarchischen Graphen auftreten können.

Der syntaktischen (Hyper-)Regel
 $\text{synh}_1 = \text{program} : \text{block}.$

läßt sich z. B. die semantische (Hyper-)Manipulationsvorschrift der Figur 8/2 zuordnen.

$semh_1 =$

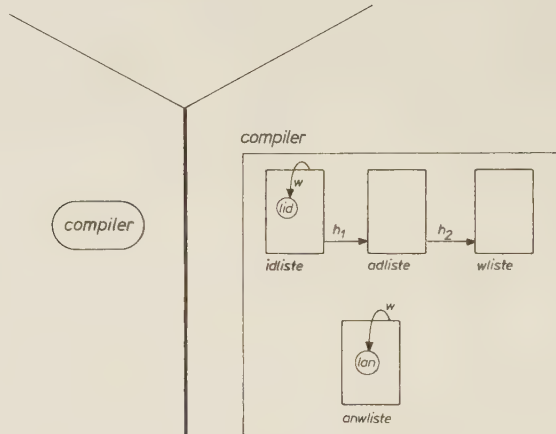


Fig. 8/2

Dieser einfache 'compiler' führt vier Listen, die *Identifikatorliste*, die *Adressenliste*, die *Werteliste*, die *Anweisungenliste*; $semh_1$ stellt seine Initialisierung dar. 'compiler' ist in $liS(semh_1)$ eine Eckenmarkierung, in $reS(semh_1)$ eine Eckenbezeichnung. Die Bezeichnungen der Ecken in der Ecke compiler erklären sich aus dem o. g.. Die Ecke $\textcircled{lid}$ soll (die i. a. oft zu ersetzende Ecke) "letzter Eintrag in die Identifikatorliste" darstellen, analog $\textcircled{lan}$; w steht für weiter (also die einfache lineare Anordnung, die sich insbesondere durch die Anwendung von $semh_2$ ergibt); h_1 und h_2 sind zwei Hilfskantenmarkierungen.

Es wird ein sehr einfaches Modell vorgestellt, um die Prinzipien der beabsichtigten semantischen Beschreibung deutlich werden zu lassen. Dabei ist klar, daß sich die Ecke 'compiler' in Figur 8/2 noch weiter verfeinern ließe, um so näher an die Darstellung eines realen Compilers zu gelangen.

Die Semantik der *Vereinbarung*

synh_2 = verweis auf ART indentifikatorvereinbarung :
ART-symbol, IDENTIFIKATOR, semikolon-symbol.
wird durch Figur 8/3 erklärt^{*)}.

Nehmen wir an, daß für ART die Metaregeln

synm_1 = ART :: verweis auf ART.

synm_2 = ART :: integer.

bestimmend sind. Zugeordnet werden die Metamanipulationsvorschriften aus Figur 8/4(a) bzw. 8/4(b)

^{*)} An dieser Stelle wird deutlich, daß man in der Definition (3/) für quasihierarchische Graphen, bzw. in der Definition eines Graphen selbst (Kap. 2.1), in irgendeiner Weise eine 'Form der Ecken' des Graphen mitführen könnte, um keine Mehrdeutigkeiten bei der graphischen Repräsentierung dieser Objekte aufkommen zu lassen. Formal ließe sich dies so beschreiben: Sei Eckf eine (endliche) Menge von *Eckenformen*. Dann ist für $e \in E(g)$ eines Graphen g emark darzustellen als eine Funktion $\text{emark}: E \rightarrow \text{Eckm} \times \text{Eckf}$. Für $\{e\} := \text{liS}(\text{semm}_1)$, Figur 8/4(a), könnte man definieren: $\text{emark}(e) = (\text{ART}, \triangle)$.

Diese feine Unterscheidung würde aber für uns eine unnötige Verkomplizierung mit sich bringen und es wird deshalb auch darauf verzichtet.

r steht für 'Referenz auf' als Kantenmarkierung in den Figuren.

$semh_2 =$

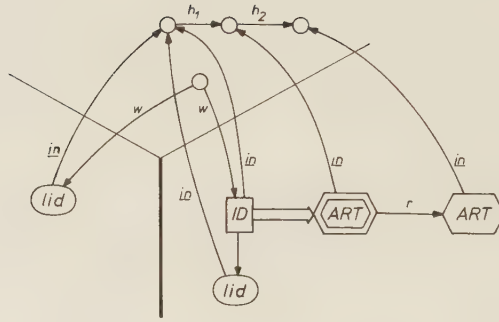


Fig. 8/3

$semm_1 =$

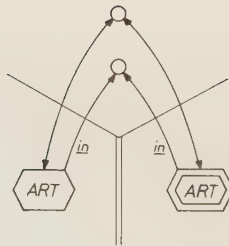


Fig. 8/4(a)

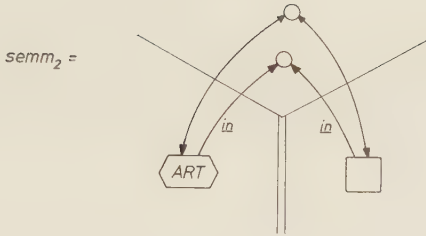


Fig. 8/4(b)

IDENTIFIKATOR sei stets nach einer noch nicht vorhandenen terminalen Zeichenkette ableitbar, zugeordnet sei $semm_3$ der Figur 8/5.
 $synm_3 = \text{IDENTIFIKATOR} :: \text{alphasequenz-symbol}.$

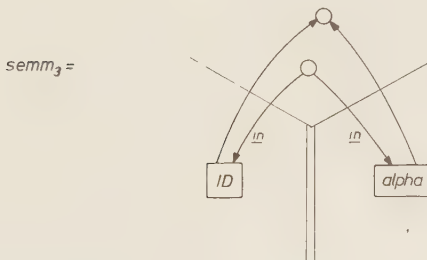


Fig. 8/5

compiler

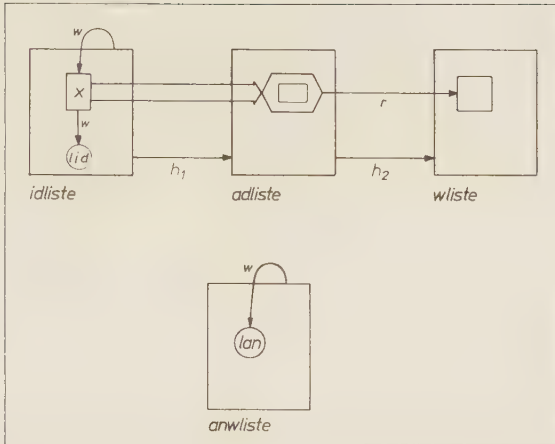


Fig. 8/7

Man sieht leicht, daß man das Weiterschieben des Zeigers (*lid*), des letzten Eintrags in die Identifikatorliste, auch durch die Vorschriften

```
synh3 = identifikatorenvereinbarungen : verweis auf
      ART identifikatorvereinbarung,
      identifikatorenvereinbarungen,
```

```
synh4 = identifikatorvereinbarungen : .
```

beschreiben könnte, mit der Semantik aus Figur 8/8. Natürlich müßte dann $\text{emark}(\text{lis}(\text{semh}_2)) = \text{vlid}$ sein. ("vorletzter Eintrag in die Identifikatorliste").

$semh_3 =$

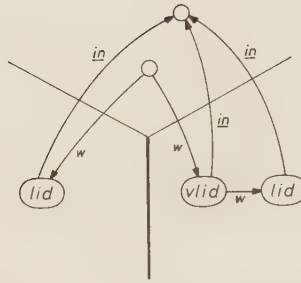


Fig. 8/8

Eine Vereinbarung

ref integer y ; ,

beschreiben durch $synh_2 [synm_1 \cdot synm_2]$ (also: $synh_2$ wird durch Anwendung von $synm_1$ und $synm_2$ (in dieser Reihenfolge) produktiv gemacht und ergibt als semantische Manipulationsvorschrift Figur 8/9.) Analog dazu würde sich der Inhalt der Ecke mit der Bezeichnung compiler verändern.

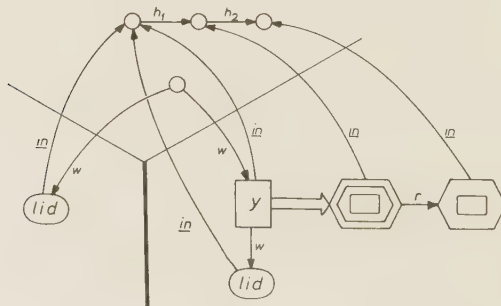


Fig. 8/9

Die Semantik eines *Blocks* ließe sich z. B. durch die Manipulationsvorschrift in Figur 8/10(a) bzw. 8/10(b) darstellen, wobei Figur 8/10(a) das Auftreten eines begin beschreibt (es tritt also eine neue Ecke in der Ecke mit der Bezeichnung *idliste* auf, die die zu definierenden Identifikatoren aufnimmt), und Figur 8/10(b) das 'Herausschieben' des Zeigers bei end beschreibt.

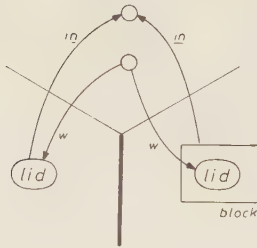


Fig. 8/10(a)

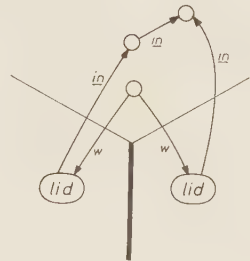


Fig. 8/10(b)

Für die Beschreibung der Semantik von *Wertzuweisungen* sind die verschiedensten Darstellungen denkbar. So sei mit Figur 8/11 die Semantik von Wertzuweisungen beim Fehlen von automatischen Artanpassungen beschrieben.

Dies bedeutet, daß sich alle Kanten übertragen und nur eine *kopier*-Kante eingesetzt wird.

Die Figur 8/11 ist ein Beispiel dafür, wie sich die Semantik der Zeichen des Semantikkalküls in der Darstellung eines Sachverhaltes niederschlagen. Nehmen wir an, wir hätten die Semantik einer Manipulationsvorschrift f so definiert, daß in einem vorliegenden Graphen der zu $lis(f)$ korrespondierende Untergraph nur dann durch $res(f)$

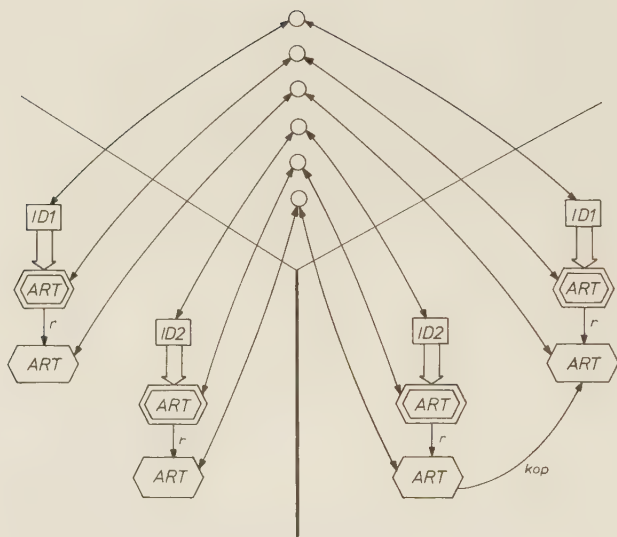


Fig. 8/11

ersetzt wird, wenn die Anschlußbedingung für jede neue Kante von f erfüllt ist, hätte man unter Vernachlässigung der in-Kanten denselben Sachverhalt einfach durch Figur 8/12 darstellen können^{*)}.

^{*)} Man beachte, daß die Figuren 8/11 und 8/12 auch Wertzuweisungen an Identifikatoren erlauben, die in parallelen Blöcken definiert sind. In Figur 8/12 wäre ein Verbot solcher Wertzuweisungen leicht zu beschreiben: Man muß nur zu einer zusätzlichen, unmarkierten Ecke in E_a von den mit ID1 bzw. ID2 markierten Ecken je eine in-Kante verlaufen lassen. Figur 8/11 hingegen müßte völlig umgestaltet werden.

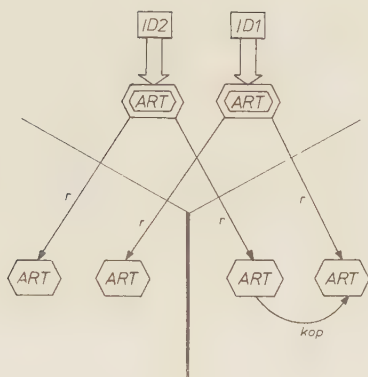


Fig 8/12

Man sieht leicht, daß die obigen Darstellungen die Wertzuweisung $x:=x$ nicht beschreiben können. Dies muß über Einträge in der Ecke anwliste geschehen mit

$\text{synh}_5 = \text{wertzuweisung} : \text{verweis auf ART ziel, ergibt symbol, (starke) ART quelle.}$

Die Klammerung von 'starke' soll bedeuten, daß mittels weiterer syntaktischer Hyperregeln die Artanpassungen bei 'starkem' Kontext zu beschreiben wären. Figur 8/13 macht einen Vorschlag für 'einfache' Wertzuweisungen.

Es ist klar, daß der Interpreter, der den Semantikgraphen nach der Programmübersetzung abarbeitet, den eigentlichen Kopiervorgang sowie die Zuordnung der Teilgraphen, die an den aus ID1 bzw. ID2 terminal abgeleiteten Ecken 'hängen', auszuführen hat.

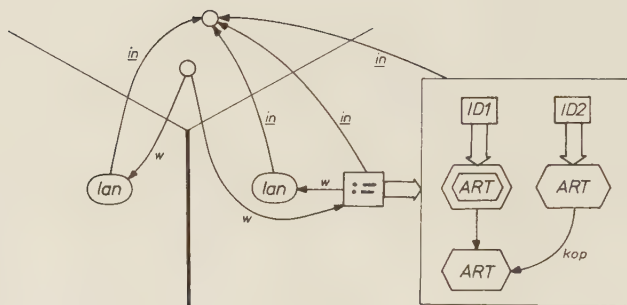


Fig. 8/13

Die Semantik von *Marken* ließe sich durch Figur 8/14 darstellen, die von *Sprüngen* z. B. durch Figur 8/15. Für *Abfragen* könnte man die gewohnten graphischen Darstellungen wählen.

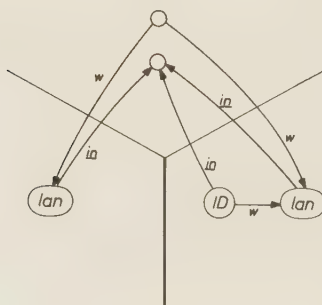


Fig. 8/14

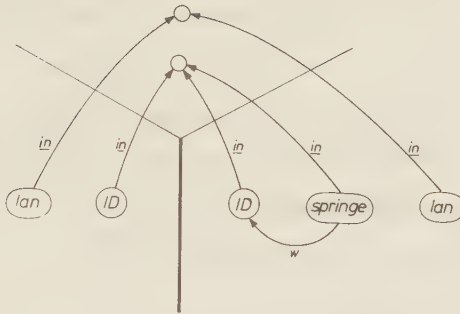


Fig. 8/15

Im folgenden wird noch ein Beispiel angegeben, wie man den Komplex der *Dereferenzierung* durch ein zweistufiges Graphmanipulationssystem beschreiben kann. Um keinen unnötigen Ballast mitschleppen zu müssen, verwenden wir andere Bezeichnungen, um anzudeuten, daß dieses Beispiel losgelöst von unserem einfachen quasihierarchischen Graphen compiler betrachtet werden soll.

Vereinfacht kann der Sachverhalt der Wertzuweisung mit automatischer Dereferenzierung so beschrieben werden: "Es ist der Inhalt derjenigen Ecke e der Quelle der Wertzuweisung in die zweite Ecke e' unter dem Identifikator des Ziels zu kopieren, deren Markierung mit der Markierung von e' übereinstimmt^{*)}". Wir drücken dies durch eine Kante (e, e', kop) mit der Markierung kop aus. Wie können wir durch ein zweistufiges Graphmanipulationssystem für hierarchische Graphen garantieren, daß stets die 'richtigen' Ecken kopiert werden?

^{*)} s. a. [34].

Wir benötigen folgende syntaktischen Hyperregeln, die [73] entnommen sind und für unseren Fall spezialisiert wurden:

$\text{synh}_6 = \text{reference to MODE assignment : reference to MODE mode IDENTIFIER, becomes symbol, MODE source.}$

$\text{synh}_7 = \text{MODE source : MODE mode IDENTIFIER.}$

$\text{synh}_8 = \text{MODE source : dereferenced to MODE mode IDENTIFIER.}$

$\text{synh}_9 = \text{dereferenced to MODE mode IDENTIFIER : reference to MODE mode IDENTIFIER.}$

$\text{synh}_{10} = \text{dereferenced to MODE mode IDENTIFIER : dereferenced to reference to MODE mode IDENTIFIER.}$

Dazu kommen folgende syntaktischen Metaregeln.

$\text{synm}_4 = \text{MODE :: reference to MODE.}$

$\text{synm}_5 = \text{MODE :: integer.}$

Für das uninteressante Metanichtterminalzeichen IDENTIFIER verzichten wir auf Ableitungsregeln sowohl in der syntaktischen wie auch in der semantischen Beschreibung.

synh_6 fordert das Vorhandensein einer mindestens einfach referenzierten Größe als linke Seite einer Wertzuweisung. Dies stellen wir wieder durch eine Ecke mit sechseckiger Berandung dar. Die Doppelkante repräsentiere eine spezielle Markierung ('besitzt') und r stellt Referenzierung dar. Es ist klar, daß der obere Teil von ∇ in Figur 8/16 zur Platzierung der Ecken von $\text{reS}(\text{semh}_6)$ in compiler dienen könnte.

$semh_6 =$

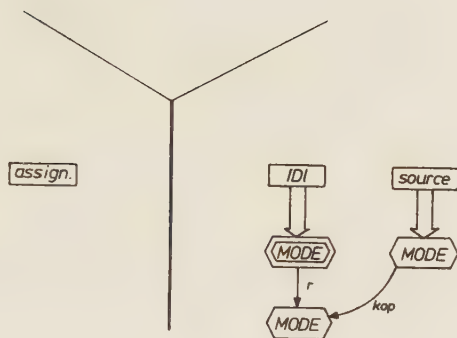


Fig. 8/16

$semh_7 =$
 $semh_8 =$

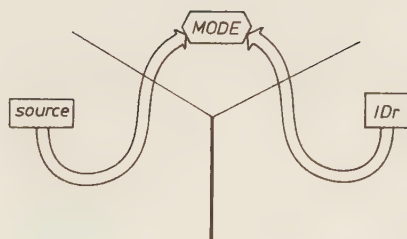


Fig. 8/17

$semh_9 =$
 $semh_{10} =$

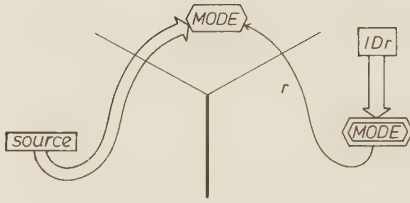


Fig. 8/18

Was in $semh_7$ bzw. $semh_8$ der Figur 8/17 geschieht, ist klar. Die Anwendung von Figur 8/18 hat den Effekt, daß die *Kopier*-kante verschoben wird.

Figur 8/19 bzw. 8/20 beschreiben die Semantik von $synm_4$ bzw. $synm_5$.

$semm_4 =$

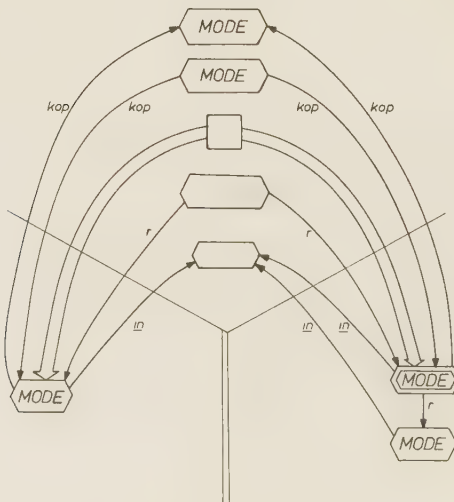


Fig. 8/19

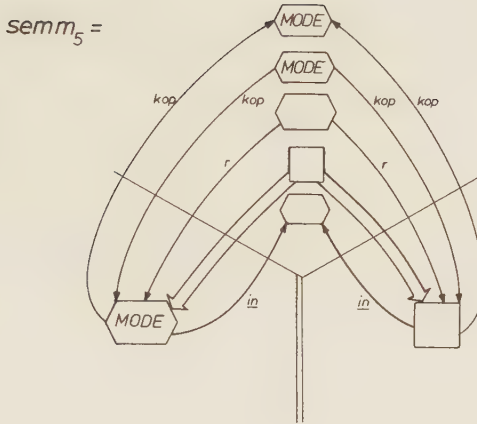


Fig. 8/20

Betrachten wir eine Wertzuweisung

$x := 1;$

wobei x von der Art ref int und 1 von der Art int ist.

Die Syntexanalyse hat dazu $synh_6[sym_5] \cdot synh_7[sym_5]$ gefunden (d. h.: Zuerst wurde $synh_6$ angewendet, nachdem sie mittels sym_5 produktiv gemacht wurde, danach $synh_7$, die ebenfalls durch sym_5 produktiv gemacht wurde.). $semh_6[semm_5]$ wird in Figur 8/21 beschrieben.

$semh_6[semm_5] =$

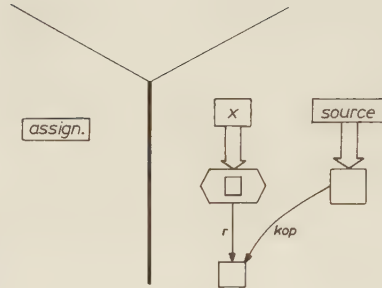


Fig. 8/21

Diese Manipulationsvorschrift wird auf eine Ecke assign. angewendet, wobei sich die rechte Seite von Figur 8/21 bildet. Figur 8/22 beschreibt $semh_7[semm_5]$.

$semh_7[semm_5] =$

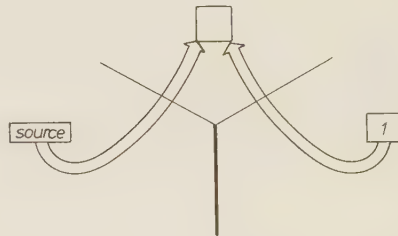


Fig. 8/22

Es gilt nun die Beziehung in Figur 8/23, die die Semantik dieser Wertzuweisung beschreibt.

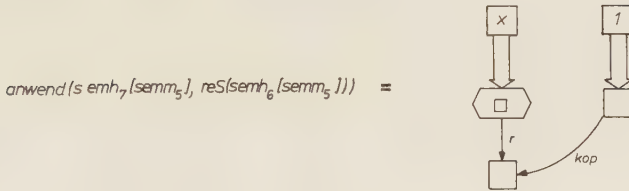


Fig. 8/23

Betrachten wir nun die Wertzuweisung

$u := v;$

wobei u von der Art ref int und v von der Art ref ref int sein soll. Die Syntaxanalyse ergibt:

$\underbrace{synh_6[sym_5]}_{\text{Figur 8/21}} \cdot \underbrace{synh_8[sym_5]}_{\text{Figur 8/24}} \cdot \underbrace{synh_{10}[sym_5]}_{\text{Figur 8/25}} \cdot \underbrace{synh_9[sym_4, sym_5]^*}_{\text{Figur 8/26}}.$
Figur 8/27

Die gebildeten semantischen Manipulationsvorschriften werden auf die Ergebnisse der vorangehenden Manipulation angewendet. Begonnen wird mit der Manipulation der Ecke assign.

*) Dieser Ausdruck beschreibt, von links nach rechts gelesen, die Reihenfolge der Anwendungen der Hyperregeln und der Metaregeln auf diese. Die Figurenangaben beschreiben die zugeordneten semantischen Manipulationsvorschriften.

$semh_8[semm_5] =$

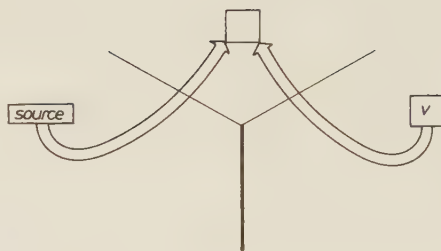


Fig. 8/24

$semh_{10}[semm_5] =$

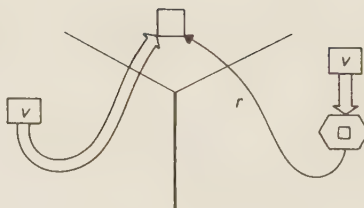


Fig. 8/25

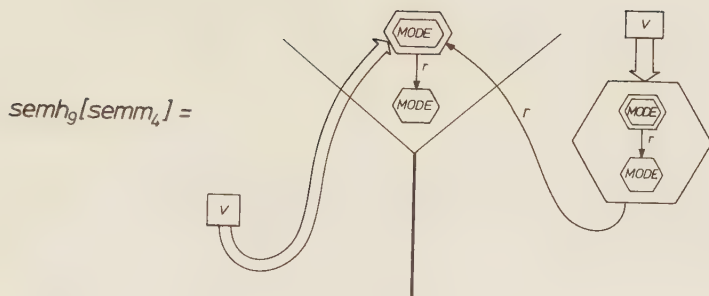


Fig. 8/26

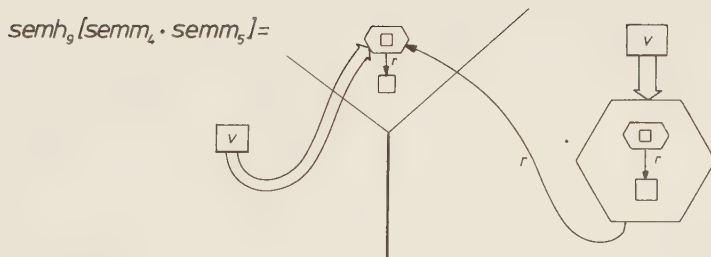


Fig. 8/27

Die schrittweise Manipulationsfolge ist in Figur 8/28 dargestellt. Man sieht auch, daß die semantischen Manipulationsvorschriften so gewählt wurden, daß in Semantik(quasihierarchischen)graphen in jeder Ecke mit Sechseckumrandung die Gestalt des 'darunter' liegenden Graphen aufgehoben wird.

In der Wertzuweisung

$u:=v;$

soll u von der Art ref ref int und v von der Art ref ref ref int sein. Die Syntaxanalyse findet die Ableitungsfolge:

$$\text{synh}_6[\text{synm}_4 \cdot \text{synm}_5] \cdot \text{synh}_8[\text{synm}_4 \cdot \text{synm}_5] \cdot \text{synh}_{10}[\text{synm}_4 \cdot \text{synm}_5] \cdot \text{synh}_9[\text{synm}_4 \cdot \text{synm}_4 \cdot \text{synm}_5]$$

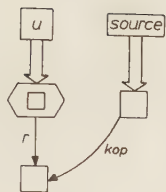
Die zugehörigen semantischen Manipulationsvorschriften sind in Figur 8/29 beschrieben. Der Einfachheit halber werden die Aktualisierungen mittels semm_5 nicht extra durchgeführt. Aus Platzgründen wurde MODE durch M abgekürzt und die Markierung r weggelassen.

Die Ableitungsfolge wird in Figur 8/30 angegeben.

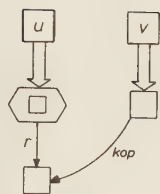
Das Ergebnis ist äquivalent zu dem, was man in [34] als gewählte Darstellung erwarten würde.

assign.

$\vdash \text{semh}_6[\text{semm}_5]$

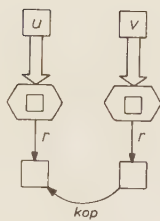


$\vdash \text{semh}_6[\text{semm}_5]$



$\vdash$

$\vdash \text{semh}_6[\text{semm}_5]$



$\vdash \text{semh}_5[\text{semm}_4, \text{semm}_5]$

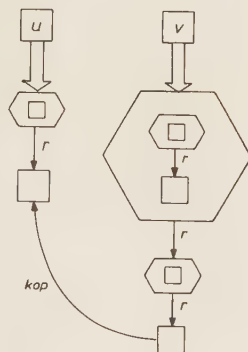


Fig. 8/28

$semh_6[semm_4, semm_5] \bullet semh_8[semm_4, semm_5] \bullet semh_{10}[semm_4, semm_5] \bullet semh_9[semm_4, semm_4, semm_4, semm_5]$

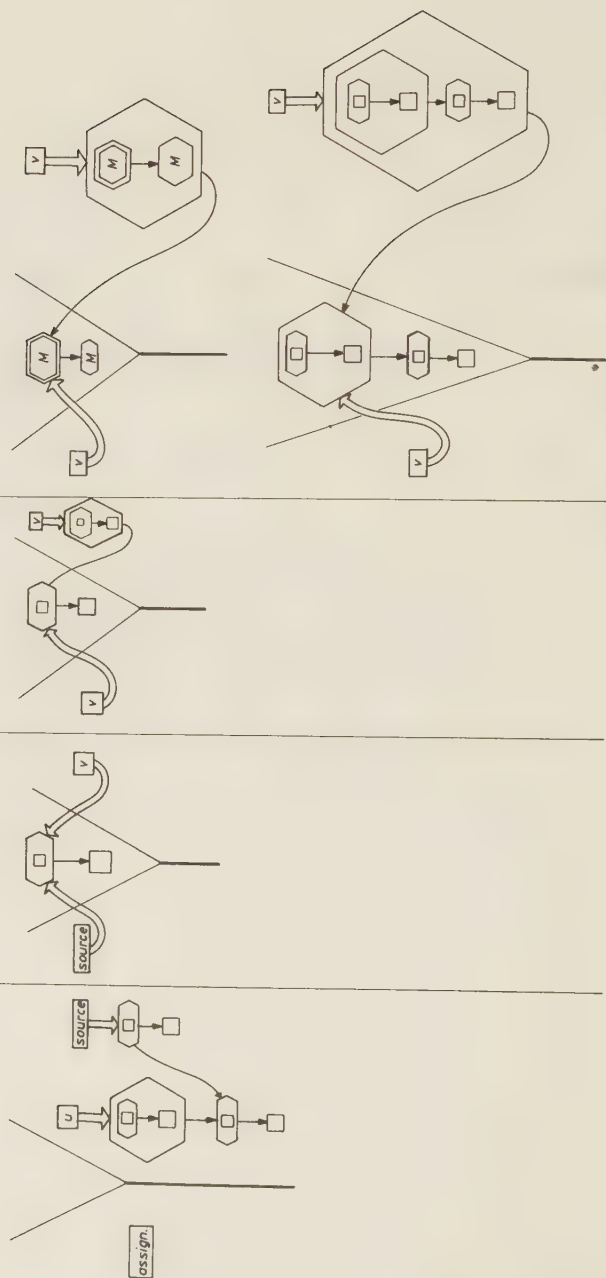


Fig. 8/29

assign.

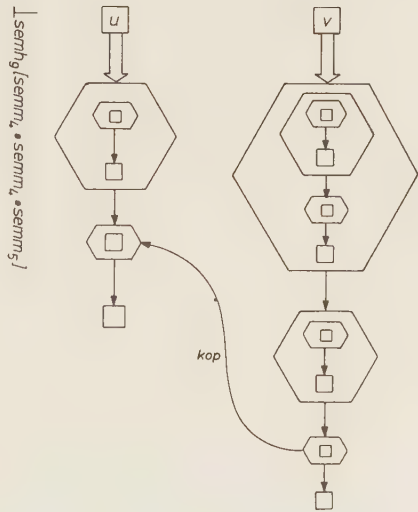
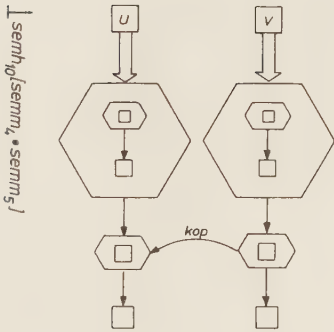
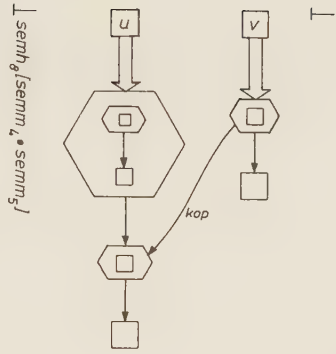
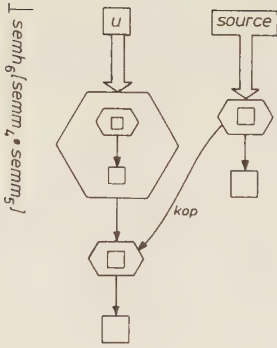


Fig. 8/30

Aus den o. g. Beispielen geht hervor, daß man ohne gro-
se Schwierigkeiten auch Flußdiagrammsprachen (s. a. [58]
oder Programmschematasprachen mittels zweistufiger Graph-
manipulationssysteme für quasihierarchische Graphen dar-
stellen kann. Ebenso ist es möglich, durch sie eine bis
auf die 'Bitebene' heruntergehende Semantikbeschreibung
auszudrücken. Es ist klar, daß der Aufwand stark von den
verwendeten Graphen abhängt, nicht zuletzt davon, wie
diese mit der Intuition des Benutzers solcher Systeme
korrespondieren.

9 WEITERE PROBLEME

Bei der Diskussion der behandelten Probleme trat eine Reihe weiterer Fragen auf. Die wichtigsten sollen im folgenden kurz zusammengefaßt dargestellt werden.

1) Theoriebezogene Aspekte

1.1) bei einfachen Manipulationsvorschriften.

1.1.1) Im Kapitel 4 sind bereits Alternativen für mögliche sinnvolle Definitionen von $\text{anwend}(f,g)$ angegeben. Sie lassen sich leicht aus den Kombinationsmöglichkeiten für die gewählten Auffassungen für die linken Seiten bzw. für die Anschlußbedingungen ableiten, entsprechend der Vorschrift, ob für die linke Seite bzw. die Anschlußbedingungen die Existenz eines Untergraphen, eines Teilgraphen oder eines Homomorphismus ^{*)} für die zu manipulierenden Graphen als Voraussetzung für die Ersetzung gefordert wird. Dies kann noch gepaart werden mit der Forderung nach dem gleichzeitigen Erfülltsein des Vorhandenseins der linken Seite *und* dem mindestens einmaligen Auftreten der Anschlußbedingungen. Interessant wäre eine Gegenüberstellung der Effekte, die ein und dieselbe Manipulationsvorschrift f bei den verschiedenen Alternativen für $\text{anwend}(f,g)$ bei vorgegebenen Graphklassen hat.

1.1.2) Eine weitere Verringerung der Bedingungen, die für eine Ersetzung erfüllt sein müssen, ist die Angabe von partiellen Graphen für die linke Seite und die Anschlußbedingungen, d. h. von Graphen, die Kanten k mit undefiniertem $\text{ae}(k)$ bzw. $\text{ee}(k)$ besitzen. Solche Manipulationsvorschriften umfassen eine Menge von 'ähnlichen' Vorschriften, die entstehen, wenn man die unvollständigen Graphen durch Hinzunahme von Ecken $e_1, e_2, \dots, e_n$

^{*)} s. a. [54]

vervollständigt, wobei $\text{emark}(e_1)$ Eckm durchläuft. Implikationen solcher Graphersetzungen werden in [64] diskutiert, allerdings im Hinblick auf einen andersgearteten Graphmanipulationsansatz.

- 1.1.3) Bislang wurde nur untersucht, wie das Ziehen der neuen Verbindungskanten unter der Bedingung des Erfülltseins der Anschlußbedingungen zu beschreiben ist. Interessant und anwendungsmotiviert ist die Frage nach den Resultaten, wenn man als Bedingung für das Ziehen der neuen Kanten das *Nicht*vorhandensein von Teilen bzw. der ganzen Anschlußbedingung fordert, was z. B. durch ein Durchstreichen der entsprechenden Kanten in der Anschlußbedingung darstellbar wäre.
- 1.1.4) Wie ändern sich die Beschreibungsmöglichkeiten, wenn man erlaubt, daß die Suche nach den zu den Anschlußbedingungen äquivalenten Teilgraphen von den Ankerecken ausgehend auch über den zur linken Seite der Manipulationsvorschrift äquivalenten Untergraphen gehen darf?
- 1.1.5) Von Interesse ist die Übertragung der klassischen Fragestellungen aus der Theorie der Formalen Sprachen auf Graphmanipulationssysteme, z. B. die Entscheidbarkeit des Wortproblems, Normierungsfragen, d. h. welche Gestalt die Anschlußbedingungen bzw. die linken Seiten von Manipulationsvorschriften erfüllen müssen, um gewisse Graphenklassen zu erzeugen, z. B. Bibäume, balancierte Bäume, vollständige Graphen usw..
- 1.1.6) Ein weites Betätigungsfeld sind Syntaxanalyseverfahren für Graphmanipulationssysteme. Es sind keine Ergebnisse diesbezüglich bekannt, außer

bei Manipulationsvorschriften mit trivialen Anschlußbedingungen.

1.2) bei zweistufigen Manipulationssystemen.

1.2.1) Die unter 1.1 genannten Fragestellungen treten auch bei der Behandlung der Zweistufigkeit auf, etwa Entscheidbarkeitsprobleme der üblichen Art, die Alternativen zu $\text{panwend}(m,h)$ bzgl. der Möglichkeiten aus 1.1.1), oder das Verwenden der 'verbotenen' Kanten aus 1.1.3), d. h. man fragt nach den Implikationen, wenn man in Metamanipulationsvorschriften doppelt durchgestrichene Kanten erlaubt und diese so interpretiert, daß eine doppelt durchgestrichene Kante auf eine einfach durchgestrichene Kante einer Hyperregel so wirkt, wie eine einfach durchgestrichene auf eine nicht durchgestrichene Kante.

1.2.2) Welche möglichst schwachen Verträglichkeitsbedingungen zwischen Meta- und Hypermanipulationsvorschriften lassen sich finden?

1.2.3) Unter welchen Bedingungen ist $\text{panwend}(m,h)$ durch eine Kette von reihenfolgenabhängigen Anwendungen von $\text{anwend}(m,h)$ ersetzbar? Ansätze zur Beschreibung solcher Sequentialisierungsmöglichkeiten finden sich in [44].

1.2.4) In Verbindung mit der Untersuchung von partiellen Graphen ist die parallele Ersetzung von attribuierten Kanten interessant.

2) Anwendungsbezogene Aspekte

2.1) In [20] wurde untersucht, wie aufwendig eine Semantikbeschreibung wird, deren zugrunde liegenden Zeichen ursprünglich Bäume sind, und die auf nicht baumartige Objekte angewendet wird. Wie ändert sich der Aufwand,

wenn man nichtrechtseindeutige Relationen darstellen will, d. h. es gibt im beschreibenden Graphen eine (oder mehrere) spezielle Kantenmarkierungen, wie z. B. in, die keine Waldstruktur besitzen (anschaulich: Ersetzungen bei sich im Inhalt überlappenden Ecken)?

- 2.2) Inwieweit erlauben die Graphmanipulationssysteme eine Beschreibung ihrer Semantik durch sich selbst (im Sinne des Schreibens eines Compilers für eine Sprache in der Sprache selbst)?
- 2.3) Für welche programmiersprachlichen Aspekte sind die einzelnen in 1.1.1) und 1.1.2) genannten Alternativen am besten geeignet? Manche Sachverhalte werden durch *Ersetzungen* in Graphen nicht adäquat beschrieben; wenn z. B. nur eine Kante zwischen zwei Ecken in einem Semantikgraphen gelöscht werden soll, müssen wir dies in unserem Kalkül so beschreiben, daß eine Manipulationsvorschrift angewendet wird, deren linke Seite aus den Ecken *mit* der betrachteten Kante besteht und deren rechte Seite nur die Ecken *ohne* diese Kante enthält. (Manipulationssysteme für partielle Graphen würden dem o. g. Sachverhalt besser entsprechen.)

Diese Aufzählung mag genügen, um einen Eindruck der Problemfülle zu vermitteln, die im Zusammenhang mit einstufigen und zweistufigen Graphenmanipulationssystemen offenbar wird.

10 VERGLEICH MIT ANDEREN ARBEITEN

Zum Abschluß soll noch ein kurzer Eindruck gegeben werden, wie sich diese Arbeit zu verschiedenen anderen aus dem Bereich der Graphersetzungssysteme und der Semantikbeschreibung verhält.

Folgende Hauptdiskussionspunkte lassen sich im vorliegenden Text finden: Der erste Teil befaßte sich mit der Beschreibung einer 'Manipulationsvorschrift', die selbst wiederum ein Graph ist. Diese Darstellung erschien deshalb notwendig, damit die Manipulation einer Manipulationsvorschrift, die wiederum Graphen manipuliert, mit demselben Mechanismus beschrieben werden kann. Dies führte über eine verallgemeinerte 'parallele Anwendung' im zweiten Teil zu 'zweistufigen Graphmanipulationssystemen'. Der dritte Teil der Arbeit beschäftigte sich mit dem Einsatz von 'Graphmanipulationssystemen in der Semantik'. Dabei wurde das Konzept der Zuordnung von semantischen zu syntaktischen Regeln erörtert, einschließlich der Beschreibung semantischer Sachverhalte durch quasihierarchische Graphen.

In bezug auf die obigen apostrophierten Diskussionspunkte soll kurz die vorliegende Arbeit mit anderen aus der Literatur bekannten verglichen werden.

10.1 ANDERE GRAPHMANIPULATIONSANSÄTZE

Für die Diskussion weiterer Graphmanipulationsansätze wollen wir uns auf zwei beschränken, die in ihrer Beschreibungsmächtigkeit die anderen umfassen. Wir werden somit den ganzen Bereich jener Ansätze außer acht lassen, die entworfen wurden, um hauptsächlich auf dem Gebiet der Mustererkennung für mehrdimensionale Zeichen, linguistische Methoden einzuführen. Dabei versucht man, den Aufbau von komplexen Bildern durch die Angabe von Anzahl und Art der darin enthaltenen 'atomaren', also als nicht weiter zerlegbar angesehenen Strukturen zu beschreiben. Diese 'picture primitives' sind allerdings meist auf sehr einfache Art miteinander zu koppeln, so daß sich das Anschluß- oder Embeddingproblem in seiner Allgemeinheit (wie für Graphen) nicht stellte, s. z. B. [67]. Eine ausführliche Literatursammlung darüber findet sich in [19].

Von Interesse, weil vergleichbar mit der Mächtigkeit des in dieser Arbeit beschriebenen Ansatzes - im folgenden *graphentheoretischer Ansatz* genannt - ist der *kategorielle* [16]*) und der *mengentheoretische* Ansatz [43].

- 1.1) Im *kategoriellen Ansatz* wird die Einbettung eines Graphen in einen Wirtsgraphen über eine Pushout-Konstruktion erklärt.

Zunächst liegt eine Manipulationsvorschrift in der Form vor, daß zu drei Graphen liS , reS und K zwei Graph(homo)morphismen $p_l: K \rightarrow liS$ und $p_r: K \rightarrow reS$ definiert sind (s. Figur 10,1/1 oberer Teil).

*) (Eine Verallgemeinerung findet sich in [54], die, für den kategoriellen Ansatz, ein Analogon zum Übergang von anwend (Definition 4/8) zur Alternative 3) auf Seite 47 darstellt.)

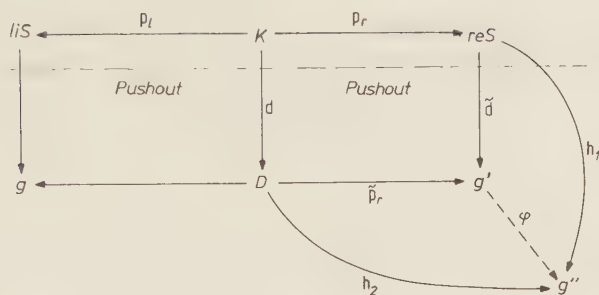


Fig. 10.1/1

Man gibt sich zwei weitere Graphen D und g' , sowie drei Homomorphismen $d:K \rightarrow D$, $\tilde{d}:reS \rightarrow g'$ und $p_r:D \rightarrow g'$ so vor, daß das Diagramm mit den Eckpunkten D , g' , reS und K kommutiert. Durch die Zusatzbedingung, daß es ein Pushout sein muß, wird garantiert, daß ein g'' , das im ebenfalls kommutativen Diagramm mit den Eckpunkten D , g'' , reS und K auftritt, (mittels φ) isomorph zu g' ist, d. h. g' ist bis auf Isomorphie eindeutig bestimmt (Figur 10.1/1).

Die explizite *Einklebungs*konstruktion läßt sich folgendermaßen angeben: Man bildet die disjunkte Vereinigung der Ecken- und Kantenmengen von D und reS , also $E(D) \sqcup E(reS)$ bzw. $K(D) \sqcup K(reS)$. In diesen Mengen geht man über zu Äquivalenzklassen, indem man die Elemente identifiziert, die unter d und p_r dasselbe Urbild haben.

Der kategorielle Ansatz erlaubt z. B., daß man Ecken, die im Sinne des graphentheoretischen Ansatzes ununterscheidbar sind, dennoch unterscheiden kann und ein Anschluß einer rechten Seite an

einen Restgraphen, wie in Figur 10.1/2 dargestellt, beschreibbar ist.

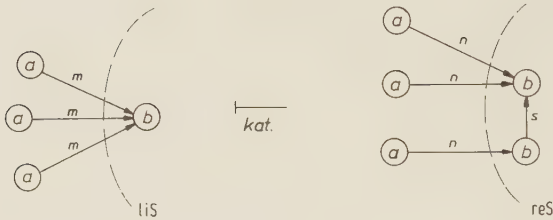


Fig. 10.1/2

Anschlüsse von rechten Seiten an Restgraphen, wie man sie im graphentheoretischen Ansatz z. B. durch eine Anschlußbedingung mit leerer Anker-eckenmenge darstellen kann, sind durch den kategoriellen Ansatz nicht nachbildbar.

- 1.2) Beim *mengentheoretischen Ansatz* besteht eine Produktion (Manipulationsvorschrift) aus zwei Teilen:

1) $liS ::= reS$

2) linearen Operatorenketten, z. B.

$$r_k = ((7, 3), (7, 4)) R_j U L_i L_i.$$

liS und reS sind Graphen. Die Ziffern im Klammerausdruck korrespondieren zu Ecken aus $E(liS)$ bzw. $E(reS)$. R bzw. L deuten die Richtungen von Kanten an, 'auslaufend' bzw. 'einlaufend', die Indizierungen ihre Markierungen.

liS wird im Wirtsgraphen im mengentheoretischen

Ansatz ebenfalls als äquivalenter Untergraph gesucht. Angenommen nun, es liegt eine Operatorenkette $l_k = (1, 2)R_i$ vor. Man sucht die zu 2 korrespondierende Ecke e_2 im Wirtsgraphen und sucht sich von dieser Ecke ausgehend alle Ecken, zu denen - das ist die Bedeutung von R - eine Kante mit der Markierung i verläuft. Von diesen Ecken - das ist die Bedeutung von l - wird zu $1 \in E(\text{reS})$ eine Kante mit der Markierung k gezogen.

Durch Kombinationen von L und R kann man beliebig in den Kontext greifen.

Der mengen- und der graphentheoretische Ansatz sind unter der Voraussetzung identisch, daß man nur Operatorenketten der Tiefe 1 (d.h. u.a. es tritt keine Kombination der Form LR oder RL auf) und entsprechend nur solche Gesamtanschlußbedingungen zuläßt, bei denen es für alle Ecken eine Kante zu einer Ecke $e \in E_1(f)$ einer Manipulationsvorschrift f gibt.

Für die allgemeinen Fälle ist momentan noch nicht ersichtlich, inwieweit sich 'graphentheoretische Manipulationsvorschriften' bzw. 'mengentheoretische Produktionen' wechselseitig ggf. über Sequenzen von Produktionen bzw. Manipulationsvorschriften nachbilden lassen.

Ein Hauptunterschied zwischen beiden Ansätzen besteht darin, daß man mittels der Operatorenkette u. U. auf bereits 'betrachtete Ecken zurückkehrt', was bei Manipulationsvorschriften ausgeschlossen ist. Die beiden Ansätze kommen sich unter diesem Aspekt näher, wenn man entweder bei Produktionen das 'Zurückkehren' verbieten würde oder Manipulationsvorschriften so anwendet, wie in Kap. 4, Seite 47/48, diskutiert: Man sucht alle *homomorphen*

Bilder von $\text{anb}(k)$ in $\text{restgraph}(g, \text{liS})$.

Nachbildungen des $\mathcal{C}$ -Operator des mengentheoretischen Ansatzes könnten über Konstruktionen durchgeführt werden, wie sie in 1.1.3) des Kap. 9 angedeutet sind.

- 2) Ein Vorzug des graphentheoretischen Ansatzes liegt wohl in der Möglichkeit, auf natürliche Weise, d. h. mit ein und demselben Formalismus, *zweistufige* Graphmanipulationssysteme einzuführen. Ein ähnlicher weiterer Ansatz ist nicht bekannt.

[26] bezieht sich auf baumartige Graphen mit trivialer Einbettung (Anschluß), so daß die Problematik der 'konsistenten Ersetzung' des graphentheoretischen Ansatzes nicht auftritt.

Für die Beschreibung von parallelen Prozessen wird in [11] ein zweistufiges Graphmanipulationssystem diskutiert. Dabei handelt es sich um die Ersetzung von Bäumen durch Bäume mit isomorpher Struktur (damit ergeben sich stets sehr einfache Anschlußbedingungen). Die Produktionen der Meta- und der Hyperebene haben die Gestalt wie in 1.2). Einige Ecken besitzen Markierungen mit Parameterlisten. Metaregeln wirken üblicherweise auf die Graphen der Hyperregeln. Die die Anschlußbedingungen beschreibenden Operator Ketten bleiben unbeeinflusst.

Hat nun eine Hyperregel h auf der linken und rechten Seite Ecken gleicher Markierung, aber mit (ggf. unterschiedlichen) Parameterlisten, und wird eine Metaregel m auf diese Ecken angewendet, so werden die Markierungen der Ecken der Hyperregeln auf folgende Weise verändert: Sei $X(P, Q)$ die Markierung mit Parameterliste der Ecke der linken Seite von m ,

und $X(a,b)$ die Markierung einer Ecke in h . Einige Ecken in der rechten Seite von m mögen mit P markiert sein. Wird m angewendet, so tritt in h' (wobei $h \xrightarrow{m} h'$) der zu P korrespondierende aktuelle Parameter, also a , auf.

- 3) Die Notwendigkeit für die Bereitstellung einer Definition für eine 'konsistente Ersetzung' bei Graphmanipulationssystemen ("panwend", s. Definition 5/7), führte über die parallele Aktualisierung ("paktual", s. Definition 5/6). Der Rückgriff auf den parallelen Ersetzungsmechanismus von Graph-LINDENMAYER-Systemen [45] erlaubte uns, die Aktualisierung von Hypermanipulationsvorschriften reihenfolgeunabhängig durchzuführen.

Die Gestalt einer Produktion in [45] entspricht der aus 1.2), wobei $|E(LiS)|=1$ vorausgesetzt wird. (Eine Verallgemeinerung des dort beschriebenen Ansatzes auf ein Analogon zu PAKTUAL (s. Definition 5/5) wäre auch möglich.) Die Vergleichbarkeit bzw. Unvergleichbarkeit des mengentheoretischen und des graphentheoretischen Ansatzes, wie in 1.2) beschrieben, schlägt durch auf die Vergleichbarkeit bzw. Unvergleichbarkeit der in [45] und in Kap. 5 beschriebenen Ansätze der parallelen Ableitung in Graph-LINDENMAYER-Systemen und der Operation "paktual" (oder auch "panwend").

Andere Ansätze zu parallelen Ableitungen in Graph-LINDENMAYER-Systemen (s. z. B. [23]) wollen wir nicht weiter behandeln, da die parallele Anwendung nur als bequemes Hilfsmittel zur Beschreibung einer reihenfolgeunabhängigen 'konsistenten Ersetzung'-Analogie für zweistufige Graphmanipulationssysteme diene und andere Möglichkeiten auch denkbar sind.

10.2 ANDERE SEMANTIKBESCHREIBUNGSANSÄTZE

Es kann im folgenden nicht darum gehen, die gesamte Literatur mit dem Stichwort "Semantik" im Titel auch nur annähernd zu erfassen. (So werden wir die weiten Bereiche der 'Programmschemata' und der 'Programmverifikation', die mit Semantikbeschreibung im eigentlichen Sinn eng verknüpft sind, nicht behandeln (s. dazu z. B. [41] bzw. [36]).) Wir beschränken uns auf die *Erwähnung der bekanntesten nichtgraphischen* und auf ein kurzes *Andiskutieren der wichtigsten graphischen Ansätze* zur Semantikbeschreibung.

Wie in Kap. 7 bereits besprochen, ist Semantikbeschreibung eine Angabe darüber, welche berechenbare Funktion einem Programm zugeordnet wird. [3] ist eine leichtfaßliche, exemplarische Darstellung, wie sich die gewöhnlichen Berechenbarkeitsbegriffe POST-Systeme, rekursive Funktionen und MARKOV-Algorithmen zur Beschreibung der Semantik von Programmiersprachen verwenden lassen, [1] stellt die Verwendbarkeit des Lambda-Kalküls und [27] die der 'axiomatischen Methode' für diesen Zweck dar.

Die in Kap. 8 an Beispielen dargestellte Zuordnung zwischen syntaktischen und semantischen Produktionen (Operationen, Manipulationen) wird ganz besonders deutlich in [30], [75] und [52]. Allerdings handelt es sich dort stets um einstufige Systeme zur Syntaxbeschreibung.

Eine weitgehend algebraische Behandlung, insbesondere der von rekursiven Programmen, findet sich z. B. in [66]. Hier wird dargestellt, wie sich aus rekursiven Programmschemata eine Funktion bestimmen läßt, die die Semantik definiert.

Gemeinsam für alle obigen Ansätze ist ihre Nähe zu den

berechenbaren Funktionen als Ziel der semantischen Beschreibung und der Versuch, die Bedeutung von *linearen* Zeichenketten wiederum durch *lineare* auszudrücken. Die Verschiedenartigkeit des Benutzerkreises einer Programmiersprache legt nahe, daß man sowohl den formalen Aspekt der Beschreibung als auch den anschaulichen zu einer Synthese bringen muß. Dieser Forderung werden zweidimensionale Ansätze wohl eher gerecht.

Wie in Kap. 1 bereits erwähnt, genügt die zweidimensionale Behandlung von programmiersprachlichen Sachverhalten in [34] keinen formalen Anforderungen.

Der Ansatz in [51] erlaubt dagegen eine formal befriedigende Darstellung des Verhaltens eines Programms bei der Abarbeitung für einfache Modelle, ebenso [5]. Besonders bei letzterem wird die Zuordnung von semantischen Graphen zu syntaktischen Produktionen wieder deutlich. Der diesen Arbeiten zugrunde liegende Graphentyp ist streng vom hierarchischen Typ. Dadurch ergeben sich Schwierigkeiten bei der Beschreibung von Sachverhalten mit nicht hierarchischer Struktur, woraus die Einführung von quasihierarchischen Graphen und ihre formale Behandlung in dieser Arbeit resultierte.

Als einen großen Fortschritt in der semantischen Darstellung durch zweidimensionale Strukturen kann man die 'Wiener Definitionssprache' (VDL) bezeichnen, s. z. B. [72]. Hier wird durch die Manipulation von Bäumen, die sowohl die Syntax wie auch den 'Berechnungszustand' repräsentieren, die Semantik eines Programms beschreiben. Eine Erweiterung der Idee ist [8] und [13]. Besonders bei letzterem wird deutlich, daß die Baumstrukturen oft nicht adäquate Darstellungen von Sachverhalten sind (s. a. [20]) und deshalb Erweiterungen nötig werden.

Da die o. g. graphischen Ansätze zur Semantikbeschreibung nur sehr eingeschränkte Graphentypen verwenden, sind dort im Vergleich zur vorliegenden Arbeit nur sehr bescheidene Mechanismen für die Manipulationen vorgesehen; dafür bringen sie ein vollständiges Interpreterkonzept zur 'Abarbeitung' (im Sinne von [38] ihres Semantikgraphen mit. Dies ist bei uns nicht geschehen. Es ist aber einsichtig, daß die 'Abarbeitung' (ebenso wie die 'Entstehung'*) durch Graphmanipulationsvorschriften beschrieben werden kann, die nun in Abhängigkeit davon, ob ein 'Muster' (also eine linke Seite) im Semantikgraphen vorliegt (z. B. ein Operationszeichen als Markierung einer Ecke) diesen verändert.

*) durch die am Ergebnis der Syntaxanalyse gesteuerte Anwendung von durch Metamanipulationsvorschriften aktualisierten Hypermanipulationsvorschriften auf eine 'semantische Startecke'.

11 SCHLUSSBEMERKUNG

Die Brauchbarkeit von mehrdimensionalen Zeichen ist unbestritten, obwohl eine Diskrepanz zwischen dem Nutzen und dem i. a. sehr großen Aufwand, der zur Speicherung und Behandlung eines Graphen in einem Rechner vonnöten ist, besteht. Graphentheoretische Modelle, wie das in dieser Arbeit vorgestellte, können ihre Brauchbarkeit erst dann generell unter Beweis stellen, wenn die Operationen der verarbeitenden Rechner nicht ausschließlich vom numerisch-sequentiellen Typ sind.

Diese Arbeit soll enden mit einem Dank an Prof. Dr. Schneider und Prof. Dr. Müller sowie an meine Kollegen W. Altmann, M. Nagl, G. Stöth, D. Weber vom Lehrstuhl für Programmiersprachen und H. Bunke vom Lehrstuhl für Mustererkennung für die Hinweise und die Bereitschaft, Diskussionspartner zu sein.

Zu danken ist auch Herrn A. Wanninger für die exakte zeichnerische Ausarbeitung der graphischen Darstellungen und Frau E. Lührs für die sorgfältige Erstellung des Manuskripts.

12 STICHWORTVERZEICHNIS

	Seite
ableitbar	32
ae() (Anfangsecke)	8
$\hat{ae}$ () (Anfangsecke)	7
AE(,) (Anfangsecke eines Wegs zwischen ... und ...)	13
aK() (alte Kanten)	27
allgemeine parallele Aktualisierung	59
anb() (Anschlußbedingung)	29
ANB() (Gesamtanschlußbedingung)	32
Anfangsecke	13
Ankereckenmenge	33
anker() (Ankereckenmenge)	33
Anschließungsproblem	4
Anschlußbedingung	29
Anschlußecke	28
Anwendung einer Manipulationsvorschrift	32
anwend(,,) (Anwendung von ... auf ... in ...)	32
$\underline{aq}$ (Graphenäquivalenz)	12
äquivalenter Unterteilgraph	32
$\underline{aq}<,>$ (Menge der $\underline{aq}$ zugrunde liegenden Abbildungen)	11
$\underline{atg}$ (äquivalenter Teilgraph)	12
$\underline{atg}<,>$ (Menge der $\underline{atg}$ zugrunde liegenden Abbildungen)	13
$\underline{aug}$ (äquivalenter Untergraph)	13
$\underline{aug}<,>$ (Menge der $\underline{aug}$ zugrunde liegenden Abbildungen)	13
$\underline{autg}$ (äquivalenter Unterteilgraph)	32
$\underline{autg}<,>$ (Menge der $\underline{autg}$ zugrunde liegenden Abbildungen)	32
A() (Anschlußecke)	28
Baumstruktur	14
bewerteter, gerichteter Graph	7
Blätter	14
Blatt() (Menge der Blätter)	14
compilerorientierte Semantikbeschreibung	96
cont() (Inhalt)	15
con(,) (Graphenkonstruktionsoperator)	10

direkt ableitbar	32
$E_a()$ (Ecken des oberen Bereichs von γ)	26
$e\acute{b}\acute{e}z()$ (Eckenbezeichnungsfunktion)	7
Eckenbezeichnungsmenge	7
Eckenmarkierungsmenge	7
Eckenmenge	7
$EE()$ (Endecke eines Wegs)	13
$ee()$ (Endeckenfunktion)	7
$\acute{e}\acute{e}()$ (Endeckenfunktion)	7
E_g (Eckenbezeichnungsmenge von g)	7
$\bar{E}_g$ (Eckenmenge von g)	7
$E()$ (Eckenbezeichnungsmenge)	8
$\bar{E}()$ (Eckenmenge)	8
Einbettungsproblem	4
Einklebekonstruktion	131
$E_l()$ (Eckenmenge der linken Seite)	26
$emark()$ (Eckenmarkierung)	8
$\widetilde{emark}()$ (Eckenmarkierung)	7
Embeddingproblem	4
$eMS(,,)$ (einstufiges Manipulationssystem)	32
Endecke	13
erhaltend	88
$E_r()$ (Eckenmenge der rechten Seite)	26
gerichteter Graph	7
gerichteter Kantenzug	13
Gesamtanschlußbedingung	32
Graph	7
Graphen, äquivalente	12
"- - homomorphe	12
"- - isomorphe	12
Graphentheoretischer Ansatz	130
$Graph(Eckm,Kam)$ (Menge der Graphen über $Eckm$ und Kam)	15
Halbkanten	60
hierarchischer Graph	15
Homomorphismus	11
<u>hom</u> (homomorph)	11
$hom<,>$ (Menge der Homomorphismen von ... nach ...)	11

Hypermanipulationsvorschrift	74
<u>in</u> (Inhaltsrelation)	21
Inhalt	15
Inhaltsfunktion	20
interpretierorientierte Semantikbeschreibung	96
<u>iso</u> (isomorph)	11
Kantenmarkierungsmenge	7
Kantenmenge	7
Kategorieller Ansatz	130
K_g (Kantenmenge von g)	7
$kmark()$ (Kantenmarkierungsfunktion)	7
$K_m()$ (Menge der Kanten mit Markierung m)	8
Konnektor, metalinguistischer	25
$K()$ (Kantenmenge)	8
linke Seite	26
$liS()$ (linke Seite)	26
Manipulationssystem	77
Manipulationsvorschrift	26
maximales Element	14
$Max()$ (Menge der maximalen Elemente)	5
mengentheoretischer Ansatz	130
Metamanipulationsvorschrift	74
m-Pfad	13
neue Kante	27
Neuschreibung	32
$neu(),$ (Neuschreibung)	32
$nK()$ (neue Kanten)	27
PAKTUAL() (allgemeine parallele Aktualisierung)	59
paktual(), (parallele Aktualisierung)	69
panwend() (parallele Anwendung)	70
parallele Aktualisierung	59
parallele Anwendung	70
PFAD(,,) (Menge der Pfade zwischen ... und ... mit ,,,)	16
primitives Element	16
Produktion	32

quasihierarchischer Graph	21
rechte Seite	26
reS() (rechte Seite)	26
Restgraph	14
restgraph(,) (Restgraph)	14
Restriktion	12
restrikt<,> (Restriktion)	12
Schlinge	13
Startecke	16
Teilgraph	12
Teilgraphanteil	31
<u>tg</u> (Teilgraph)	12
tg<,> (Menge der <u>tg</u> zugrunde liegenden Abbildungen)	12
T() (Teilgraphanteil)	31
<u>ug</u> (Untergraph)	12
ug<,> (Menge der <u>ug</u> zugrunde liegenden Abbildungen)	12
ungerichteter Kantenzug	13
Untergraph	12
Untergraphanteil	31
Unterteilgraph	32
<u>verträq</u> (verträglich)	73
Verträglichkeit	73
Waldstruktur	14
WEG(,) (Menge der Wege zwischen ... und ...)	13
Weg	13
zMS (zweistufiges Manipulationssystem)	74
zMSq (zweistufiges Manipulationssysteme für quasi- hierarchische Graphen)	88
Zusammenschließungsproblem	4
zweistufiges Manipulationssystem	74
$\backslash, \cdot$ (metalinguistischer Konnektor)	25/71
$\vdash, \vdash^*$ (Ableitung)	59

1.3 L I T E R A T U R V E R Z E I C H N I S :

- [1] ABDALI, S.K.: "A Lambda-Calculus Model of Programming Languages, I, II", Journal of Computer Languages, Vol. 1, 1976.
- [2] ABE, N./MIZUMOTO, M./TOYODA, J.-I./TANAKA, K.: "Web Grammars and several Graphs", Journal of Computer and System Sciences, Vol. 7, 1973.
- [3] AUGUSTIN, H.: "Methoden zur Formalisierung der Semantik von Programmiersprachen", Mitteilungen der Gesellschaft für Mathematik und Datenverarbeitung, Bonn, 1972.
- [4] BACKES, S.: "Was ist syntaxgesteuerte Übersetzung", Elektronische Rechenanlagen, Heft 5, 1975.
- [5] BASILI, V.R./TURNER, A.J.: "A Hierarchical Machine Model for the Semantics of Programming Languages", ACM SIGPLAN Notices, Vol. 8, No. 11, 1973.
- [6] BEKIC, H./BJØRNER, D./HENHAPL, W./JONES, C.B./LUCAS, P.: "A Formal Definition of a PL/I Subset, I, II", IBM Technical Report No. 25.139, Wien, 1974.
- [7] BENSON, D.B.: "Syntax and Semantics: A Categorical View", Information and Control 17, 1970.
- [8] BJØRNER, D.: "META-IV. Reference Manual", IBM Laboratory, Wien, 1975.
- [9] BOCHMANN, G.: "Semantic Evaluation from Left to Right", Communications of the ACM, Vol. 19, No. 2, 1976.
- [10] BOEHM, C. (Ed.): "Lambda-Calculus and Computer Science Theory", Lecture Notes in Computer Science, Vol. 37, Springer Verlag, Berlin, 1975.
- [11] BUNKE, H.: "Semantikbeschreibung von Programmiersprachen mit expliziten Parallelismen", internes Kolloquium, Erlangen, 1977.
- [12] CHEN, W.-K.: "Applied Graph Theory", North-Holland Publ. Co., Amsterdam, 1971.
- [13] ČULIK, K.II: "A Model for the Formal Definition of Programming Languages", Department of Applied Analysis and Computer Science, Report CSRR 2065, Waterloo, Canada, 1972.
- [14] ČULIK, K.II/LINDENMAYER, A.: "Parallel Rewriting on Graphs and Multidimensional Development", Rept. CS-74-22 Theoret. Biology Group, Utrecht, 1974.
- [15] DENERT, E.: "PLAN2D. Konzept und Syntax einer zweidimensionalen Programmiersprache", Dissertation, Fachbereich Kybernetik, Berlin, 1975.

- [16] EHRIG,H./PFENDER,M./SCHNEIDER,H.J.: "Graph-Grammars: An Algebraic Approach", SWAT-Conference of IEEE Computer Society, 1973.
- [17] ENGELER,E.(Ed.): "Symposium on Semantics of Algorithmic Languages", Lecture Notes in Mathematics, Vol. 188, Springer Verlag, Berlin, 1971.
- [18] FRANCK,R.: "PLAN2D-Syntaxanalyse von Präzedenz-Graph-Grammatiken, Dissertation, Fachbereich Kybernetik, Berlin, 1975.
- [19] FU,K.S.: "Syntactic Methods in Pattern Recognition", Academic Press, New York, N. Y., 1974.
- [20] GOEB,H.: "Semantische Beschreibung listenverarbeitender Algorithmen durch eine Verallgemeinerung der Wiener Definitionssprache", Diplomarbeit am Lehrstuhl für Programmiersprachen, Erlangen, 1973.
- [21] GOGUEN,J.A.Jr.: "On Homomorphismus, Correctness, Termination, Unfoldments, and Equivalence of Flow Diagram Programs", Int. Journal of Computation and System Sciences, 1974.
- [22] GREIBACH,S.A.: "Theory of Program Structures, Schemes, Semantics, Verification", Springer Verlag, Berlin, 1975.
- [23] GRÖTSCH,E.: "Vergleichende Studie über parallele Ersetzung markierter Graphen", in "Graphersetzungssysteme: Zwei Beiträge zu Theorie und Implementierung", Arbeitsbericht des Inst. f. Math. Masch. und Datenverarbeitung, Bd. 9, No. 6, 1976.
- [24] HANTLER,S.L./KING,J.C.: "An Introduction to Proving the Correctness of Programs", ACM Computing Surveys, Vol. 8, No. 3, 1976.
- [25] HÄNDLER,W./HOTZ,G./LANGMAACK,H.(Eds.): "Tagung über Formale Sprachen und Programmiersprachen, Oberwolfach", Mitteilungen der Gesellschaft für Mathematik und Datenverarbeitung, Bonn, 1971.
- [26] HESSE,W.: "Vollständige formale Beschreibung von Programmiersprachen mit zweischichtigen Grammatiken", Bericht 7623 der Technischen Univ. München, 1976.
- [27] HOARE,C.A.R./LAUER,P.E.: "Consistent and Complementary Formal Theories of the Semantics of Programming Languages", Acta Informatika 3, 1974.
- [28] Hoare,C.A.R./WIRTH,N.: "An Axiomatic Definition of the Programming Language PASCAL", Eidg. Techn. Hochschule, Bericht d. Fachgr. Computer-Wissenschaften, Nr. 6, 1972.
- [29] KAMLAH,W./LORENZEN,P.: "Logische Propädeutik", Bibliographisches Institut, Mannheim, 1967.
- [30] KNUTH,D.E.: "Semantics of Context-Free Languages", Mathematical Systems Theory, Vol. 2, No. 2, 1968.
- [31] KNUTH,D.E.: "Semantics of Context-Free Languages: Correction", Mathematical Systems Theory, Vol. 5, No. 1, 1971.

- [32] LEE, J.A.N.: "Computer Semantics", van Nostrand Reinhold, New York, N. Y., 1972.
- [33] LINDENMAYER, A./ROZENBERG, G. (Eds.): "Automata, Languages, Development", North Holland Publ. Co., Amsterdam, 1976.
- [34] LINDSAY, C.H./van der MEULEN, S.G.: "Informal Introduction to ALGOL 68", North Holland Publ. Co., Amsterdam, 1971.
- [35] MANNA, Z./VUILLEMIN, J.: "Fixpoint Approach to the Theory of Computation", in: Nivat M. (Ed.): "Automata, Languages and Programming", North Holland, Amsterdam.
- [36] MARCOTTY, M./LEDGARD, H.F./BOCHMANN, G.V.: "A Sampler of Formal Definitions", ACM Computing Surveys, Vol. 8, No. 2, 1976.
- [37] MAURER, H.: "Theoretische Grundlagen der Programmiersprachen", Bibliographisches Institut, Mannheim, 1969.
- [38] MCCARTHY, J.: "Towards a Mathematical Science of Computation", Proc. IFIPS 1962, North Holland Publ. Co., Amsterdam, 1962.
- [39] MESCHKOWSKI, H.: "Richtigkeit und Wahrheit in der Mathematik", Bibliographisches Institut, Mannheim, 1976.
- [40] MONTANARI, U.G.: "Separable Graphs, Planar Graphs, and Web Grammars", Information and Control 16, 1970.
- [41] MÜLLER, H.: "Programmschemata mit Prozeduren", Arbeitsbericht des Inst. f. Math. Masch. und Datenverarbeitung, Erlangen, 1973.
- [42] MÜLLER, H.: "Diskrete Algebraische Strukturen II", Arbeitsbericht des Inst. f. Math. Masch. und Datenverarbeitung, Bd. 8, No. 2, Erlangen, 1975.
- [43] NAGL, M.: "Formale Sprachen von markierten Graphen", Dissertation, Arbeitsbericht des Inst. f. Math. Masch. und Datenverarbeitung, Bd. 7, Nr. 4, Erlangen, 1974.
- [44] NAGL, M.: "On the Relation between Graph Grammars and Graph Lindenmayer-Systems", in: "Graph Rewriting Systems II, Three Papers on Theory, Application, and Implementation", Arbeitsbericht des Inst. f. Math. Masch. und Datenverarbeitung, Bd. 9, No. 1, Erlangen, 1976.
- [45] NAGL, M.: "On a Generalization of Lindenmayer-Systems to Labelled Graphs", in [33].
- [46] NAUR, P. (Ed.): "Revised Report on the Algorithmic Language ALGOL 60", Numerische Mathematik, 1963.
- [47] NEUHOLD, E.J.: "The Formal Description of Programming Languages", IBM System Journal, No. 2, 1971.
- [48] PAGAN, F.G.: "On Interpreter-Oriented Definitions of Programming Languages", The Computer Journal, Vol. 19, Nr. 2.

- [49] PARK,D.: "Fixpoint Induction and Proofs of Program Properties",
Man Machine Intelligence 5.
- [50] PFALTZ,J.L.: "Web Grammars and Picture Description", Computer
Graphics and Image Processing, 1972.
- [51] PRATT,T.W.: "A Hierarchical Graph Model of the Semantics of
Programs", Proc. AFIPS, Summer Joint Comp. Conference, 1969.
- [52] PRATT,T.W.: "Pair Grammars, Graph Languages and String-to-Graph
Translations", Journal of Computer and System Sciences 5, 1971.
- [53] RAIN,M.: "On Compilers as Language Definitions" Proc. 1. IFIP
wg. 2.4 on MOHLL, La Grande Motte, 1974.
- [54] ROSEN,B.: "Deriving Graphs from Graphs by Applying a Production",
Acta Informatica, 4, 1975.
- [55] ROSENFELD,A./MILGRAM,D.L.: "Web Automata and Web Grammars",
Machine Intelligence 7, Edinburgh, 1972.
- [56] RUSTIN,R.(Ed.): "Formal Semantics of Programming Languages",
Courant Comp. Science Symposium 2, Prentice Hall, Englewood
Cliffs, N. J., 1972.
- [57] SALOMAA,A.: "Formal Languages", Academic Press, New York, N.Y., 1973.
- [58] SCHNEIDER,H.J.: "Chomsky-Systeme für partielle Ordnungen", Ar-
beitsbericht des Inst. f. Math. Masch. und Datenverarbeitung,
Bd. 3, No. 3, Erlangen, 1970.
- [59] SCHNEIDER,H.J.: "Chomsky-Like Systems for Partially Ordered Symbol
Sets", Bericht 1971/2/2, Technische Univ. Berlin, Informations-
verarbeitung II, Berlin, 1971.
- [60] SCHNEIDER,H.J.: "A Necessary and Sufficient Condition for Chomsky-
Produktions over Partially Ordered Symbol Sets", Lect. Notes in
Economics and Mathematical Systems, Vol. 78, Springer Verlag,
Berlin, 1972.
- [61] SCHNEIDER,H.J.: "Syntax-Directed Description of Incremental
Compilers", Lect. Notes in Comp. Science, Springer Verlag, Berlin 1974.
- [62] SCHNEIDER,H.J.: "Algorithmische Sprachen I", Vorlesungsmanuskript,
Erlangen, 1976.
- [63] SCHNEIDER,H.J.: "Conceptual Data Base Description Using Graph-
Grammars", Applied Computer Science, Vol. 4, Carl Hanser Verlag,
München 1976.
- [64] SCHNEIDER,H.J./EHRIG,H.: "Grammars on Partial Graphs", Acta
Informatica 6, 1976.
- [65] SCHWARTZ,J.T.: "Semantic Definition Methods and the Evolution of
Programming Languages", in: [55]
- [66] SCOTT,D.S.: "The Lattice of Flow Diagrams", in: [17]

- [67] SHAW,A.C.: "A Formal Picture Description Scheme as a Basis for Picture Processing Systems", Information and Control, Vol. 14, 1969.
- [68] STEEL,T.B.Jr.(Ed.): "Formal Language Description Languages for Computer Programming", North Holland Publ. Co., Amsterdam, 1966.
- [69] STRENG,W.: "PLAN2D-Semantik einer zweidimensionalen Programmiersprache", Dissertation, Fachbereich Kybernetik, Berlin, 1975.
- [70] TENNENT,R.D.: "The Denotational Semantics of Programming Languages", Communications of the ACM, Vol. 19, No. 8, 1976.
- [71] WEBER,J.: "Programmtransformationen mit attribuierten Transformationsgrammatiken", Bericht 7604 der Technischen Univ. München, 1976.
- [72] WEGNER,P.: "The Vienna Definition Language", Computing Surveys, Vol. 4, No. 1, 1972.
- [73] van WIJNGAARDEN,A.(Ed.): "Report on the Algorithmic Language ALGOL 68", Springer Verlag, Berlin, 1970.
- [74] van WIJNGAARDEN,A./MAILLOUX,B.J./PECK,J.E.L./KOSTER,C.H.A./SINTZOFF,M./LINDSEY,C.H./MEERTENS,L.G.L.T./FISKER,R.G.: "Revised Report on the Algorithmic Language ALGOL 68", Springer Verlag, Berlin, 1976.
- [75] WIRTH,H./WEBER,H.: "EULER: A Generalization of ALGOL, and its Formal Definition, I, II", Communications of the ACM, Vol. 9, No. 1 and 2, 1966.

Bisher sind folgende Arbeitsberichte des Instituts für
Mathematische Maschinen und Datenverarbeitung erschienen :
=====

Band 1

=====

- NR. 1 Schmitt, A. :
Theorie der nichtdeterministischen und unvoll-
ständigen Mealy-Automaten.
(Januar 1967)
- NR. 2 Göring, W., Henning, W., Rasche, W. :
Untersuchungen zur Kommunikation und Mustererkennung,
Teil 1: Übersicht.
(Juli 1967)
- NR. 3 Namneck, P. :
Untersuchungen zur Kommunikation und Mustererkennung,
Teil 2: Über Verfahren zur syntaktischen Analyse von
Sätzen formaler Sprachen.
(Juli 1967)
- NR. 4 Eggers, B. :
Untersuchungen zur Kommunikation und Mustererkennung,
Teil 3: Möglichkeiten der automatischen Erkennung
und selbständigen Korrektur von syntaktischen Fehlern
in formalen Sprachen.
(Januar 1968)
- NR. 5 Schneider, H. J. :
Programmierung numerischer Probleme.
(Oktober 1967)
- NR. 6 Wolf, F. :
Untersuchungen über den Vorhersagefehler bei
stochastischen Automaten.
(Februar 1968)
- NR. 7 Klar, R. :
Rechenautomaten 1.
(April 1968)
- NR. 8 Schmitt, A., Fehling, P. :
Rechenautomaten 2.
(November 1968)
- NR. 9 Weiss, S. :
Untersuchungen zur Kommunikation und Mustererkennung,
Teil 4: Ein Programmsystem zum Testen von Mustererken-
nungsverfahren unter Verwendung eines Sichtgerätes.
(Juli 1968)
- NR. 10 Dokumentation über die Arbeit des Instituts in den
Jahren 1966 bis 1968.
(Januar 1969)

Band 2
=====

- NR. 1 Vollmar, R. :
Über einen Automaten mit Speicherband.
(Januar 1969)
- NR. 2 Händler, W. :
Design Philosophy for Computer Systems.
(März 1969)
- NR. 3 Brackmann, U., Händler, W., Lange, M., Rasche, W.
Wolf, H. P. :
Beiträge zur Kommunikation Mensch - Maschine.
(April 1969)
- NR. 4 Haferkorn, J., Schmitt, A., Seidel, R., Wolf, F. :
Rechnergestützter Unterricht.
(April 1969)
- NR. 5 Böhme, J. F., Göring, W. :
Zwei Beiträge zur Mustererkennung.
(August 1969)
- NR. 6 Köhl, J. :
Datenreduktion zur Speicherung ebener Figuren.
(April 1969)
- NR. 7 Haferkorn, J., Henning, W., Rasche, W. :
Ein bildschirmorientiertes Betriebssystem für die
PDP 7.
(Mai 1969)
- NR. 8 Saffert, J. :
Mikroprogramm-Steuerwerke.
(Juni 1969)
- NR. 9 Spies, P. P. :
Rechnerstrukturen und Ablaufgeschehen in Rechen-
automaten - Analysis und Simulation.
(September 1969)
- NR. 10 Händler, W., Wolf, F. :
Bericht über eine USA-Reise - Aktuelles aus dem
Gebiet der Informatik.
(September 1969)
- NR. 11 Dokumentation über die Arbeit des Instituts im
Jahre 1969.
(Januar 1970)

Band 3
=====

- NR. 1 Schmitt, A. :
Bildschirme und Kommunikation Mensch - Maschine.
(Februar 1970)
- NR. 2 Böhme, J. F. :
Experimente mit stochastischen Automaten.
(Juli 1970)
- NR. 3 Schneider, H. J. :
Chomsky-Systeme für partielle Ordnungen.
(August 1970)
- NR. 4 Haas, G. :
Programme zur dialogischen Logik.
(Oktober 1970)
- NR. 5 Weicker, R. :
Erkennung von kontextfreien Sprachen durch
Tabulator-Turingmaschinen.
(Juli 1970)
- NR. 6 Schmitt, A. :
Optimale Vorhersage der Ausgabe stochastischer
Automaten über lange Zeiten.
(Februar 1971)
- NR. 7 Dokumentation über die Arbeit des Instituts im
Jahre 1970.
(Februar 1971)

Band 4

=====

- NR. 1 König, R. :
Reduktion und Zerlegung von Automaten.
(Juni 1971)
- NR. 2 Klar, R. :
Messung von Rechneraktivitäten.
(Juli 1971)
- NR. 3 Augustin, U. :
A Note on Arbitrary Varying Channels with
Noiseless Feedback.
A Note on Channels with Finite Memory.
(August 1971)
- NR. 4 Brüggmann, H. W. :
SIDIPRO, Dialogsprache und Echtzeit-Übersetzer
für die Programmierung eines Hochleistungs-
Sichtgerätes.
(August 1971)
- NR. 5 Händler, W. :
Strukturen von Computer-Graphics Systemen.
(Oktober 1971)
- NR. 6 Henning, W. :
Programmierung eines graphischen Sichtgerätes.
(April 1972)
- NR. 7 Dokumentation über die Arbeit des Instituts im
Jahre 1971.
(April 1972)

Band 5

=====

- NR. 1 Steierer, H. F. :
Algorithmen zur Zustandszerlegung von Automaten
unter besonderer Berücksichtigung der graphen-
theoretischen Aspekte.
(März 1972)
- NR. 2 Händler, W., Weizenbaum, J. :
Display Use for Man-Machine Dialog.
(September 1972)
- NR. 3 Dürre, K. :
Erfahrungen mit dem Programmiersystem LEKTOR-1
für Programmierten Unterricht.
(September 1972)
- NR. 4 Vollmar, R. :
SICELA - ein Interpretierer zur Simulation von
Automatennetzen.
(September 1972)
- NR. 5 Spies, P. P. :
Informatik III und Informatik IV.
Vorlesungsskriptum
(Oktober 1972)
- NR. 6 Görz, G. :
Die Sprache GEDANKEN als Beitrag zur syntaktischen
und semantischen Definition von Programmiersprachen.
(Oktober 1972)
- NR. 7 Buchzik, H. :
Maschinen-erzeugte Zufallszahlen.
(Dezember 1972)
- NR. 8 Dokumentation über die Arbeit des Instituts im
Jahre 1972.
(April 1973)

Band 6

=====

- NR. 1 Domesle, R. :
Vergleich von top-down- und bottom-up-
Kellerautomaten.
(Dezember 1972)
- NR. 2 Brüggmann, H. W. :
Zugriffsstrategien bei Rotationsspeichern.
(Juli 1973)
- NR. 3 Fachgespräch über mehrdimensionale Sprachen.
Erlangen, 14.-15. Juni 1973
- NR. 4 Händler, W. :
Bericht über einen USA-Aufenthalt 1972/1973
Aktuelles aus dem Gebiet der Informatik.
The Concept of Macropipelining.
(Mai 1973)
- NR. 5 Müller, H. :
Diskrete algebraische Strukturen.
(September 1973)
- NR. 6 Müller, H. :
Programmschemata mit Procedures.
(Oktober 1973)
- NR. 7 Leeb, K. :
Pascal Theorie.
(Dezember 1973)
- NR. 8 Chatterji, B. N. :
Two-stage Reinforcement Learning System.
An Algorithm for Economical Storage in Dynamic
Programming Approach to Pattern Recognition.
(Dezember 1973)
- NR. 9 Dokumentation über die Arbeit des Instituts im
Jahre 1973.
(Januar 1974)

Band 7
=====

- NR. 1 Eichmeier, P., Goeb, H. :
Anwendungsbeispiele von H-Graphen und VDL
zur semantischen Beschreibung.
(April 1974)
- NR. 2 Händler, W. :
Unconventional Computation by Conventional
Equipment.
(Mai 1974)
- NR. 3 Keramidis, S. :
Ein Beitrag zur Theorie der Prozesssysteme.
(August 1974).
- NR. 4 Nagl, M. :
Formale Sprachen von markierten Graphen.
(August 1974).
- NR. 5 Göttler, H. :
ALGOL 68 - Eine exemplarische Einführung.
(November 1974)
- NR. 6 Altmann, W. :
Konstruktionen zur Erweiterbarkeit von Sprachen.
(November 1974)
- NR. 7 Bunke, H. :
Beschreibung eines syntaxgesteuerten inkrementellen
Compilers durch Graph-Grammatiken.
(Dezember 1974)
- NR. 8 Jahresbericht 1974 der Informatik.
(März 1975)

Band 8

=====

- NR. 1 Ehrig, H., Nagl, M., Schneider, H. J. :
Three Papers on Graph Rewriting Systems and their
Applications.
(Januar 1975)
- NR. 2 Müller, H. :
Diskrete algebraische Strukturen II.
(April 1975).
- NR. 3 Klar, R., Wichmann, H. :
Mikroprogrammierung.
(Juli 1975)
- NR. 4 Weicker, R. :
Turingmaschinen mit assoziativem Speicherzugriff.
(Juli 1975)
- NR. 5 Schneider, H. J. :
Lexikon zur Programmierung I (ALGOL 60).
(Juli 1975)
- NR. 6 Woitok, R. :
Ein neuer Interpreter-Algorithmus für LISP
und seine Korrektheit.
(Juli 1975)
- NR. 7 Luft, A. :
Ein auf der Floyd/Hoare'schen Zusicherungsmethode
aufbauendes Verifikationsverfahren und seine
Begründung.
(August 1975)
- NR. 8 Rohrer, H. :
A Supervised Network of Adaptive Automata for
Pattern Recognition.
Drewes, B. :
Implementation of a Structural Pattern Recognition
System.
(Dezember 1975)
- NR. 9 Klar, R., Schreiber, H. :
Messungen mit dem ZÄHLMONITOR II.
(Dezember 1975)
- NR. 10 Jahresbericht 1975 der Informatik.
(März 1976)

Band 9

=====

- NR. 1 Brendel, W., Frühauf, T., Nagl, M. :
Graph Rewriting Systems II.
Three Papers on Theory, Application, and Implementation.
(Februar 1976)
- NR. 2 Brüggmann, H. W., Rudert, A., Lehr, H. :
EBIS
Strukturierte Bildprogrammierung auf höherer Ebene.
(Februar 1976)
- NR. 3 Hofmann, W., Wichmann, H. :
Konzept der Vertikalverarbeitung.
Implementierung und Analyse vertikaler Algorithmen
auf einem mikroprogrammierten Rechner.
(Mai 1976)
- NR. 4 Jordan, H. F., Smith, B. J. :
The Hardware-Software Interface.
(Juni 1976)
- NR. 5 Gernoth, B., Schmidt, B. :
Das Modell MEM-SIM.
Beschreibung eines Simulationsmodells zur Unter-
suchung einer Rechenanlage mit einer symmetrischen
Mehrprozessorkonfiguration und privaten Speichern.
(Oktober 1976)
- NR. 6 Grötsch, E. :
Vergleichende Studie über parallele Ersetzung
markierter Graphen.
Jahn, B. :
Eine Studie zur Implementierung von Graph-Grammatiken
unter Verwendung von Assoziativverarbeitung.
(Dezember 1976)
- NR. 7 Mertens, P., Neuwirth, W., Purucker, G.,
Schaller, K., Schmitt, W. :
Verknüpfung von Daten- und Methodenbanken, dargestellt
am Beispiel der Analyse von Marktforschungsdaten.
(Dezember 1976)
- NR. 8 Parallelismus in der Informatik.
Wissenschaftliches Kolloquium aus Anlass des Zehnjährigen
Bestehens des IMMD, Erlangen, 10./11. Juni 1976.
(Dezember 1976)
- NR. 9 Jahresbericht 1976 der Informatik.
(März 1977)

Band 10
=====

- NR. 1 Hoffmann, W. :
Modelle und Analysen von Teilnehmerrechensystemen,
angewendet auf das Teilnehmersystems RESPOND unter
CD 3300 MASTER.
(Februar 1977)
- NR. 2 Körber, K., Werzinger, G. :
Das Modell SIM-LAB.
Simulation eines klinisch-chemischen Laboratoriums.
(März 1977)
- NR. 3 Altmann, W., Weber, D. :
Programmierungsmethodik.
(März 1977)
- NR. 4 Maehle, E. :
Fehlertolerante Rechnerstrukturen.
(April 1977)
- NR. 5 Beth, T. :
Kombinatorik: Theorie der Vertretersysteme.
(April 1977)
- NR. 6 Mertens, P. :
Einige Hinweise zur technischen Gestaltung von
Bildschirmdialogen.
(Mai 1977)
- NR. 7 Jäpel, D. :
Ein Begriffssystem zur Formulierung von Schutzaspekten
und seine Anwendung auf Programmsysteme.
(Juni 1977)
- NR. 8 Brendel, W., Grötsch, E., Nagl, M., Schneider, H.-J. :
Graphersetzungssysteme, Beiträge zur Theorie, Anwen-
dung und Implementierung.
(Juni 1977)
- NR. 9 Rüdell, W. :
Darstellung und Vergleich abstrakter Modelle zur
Beschreibung geschützter Systeme.
(Juli 1977)
- NR. 10 Bolch, G., Gall, R., Grassl, M., Schmidt, B. :
Modelle für Trommelspeicher.
(Juli 1977)
- NR. 11 v. Eichhorn, K. :
Steiner Systeme, Codes und affine Räume.
(Juli 1977)

LEBENS LAUF

29.Dez.1946	in Bamberg geboren, Eltern: Greta Göttler, geb. Müller Hans Göttler,
Sept.1953-Aug.1957	Luitpold-Volksschule in Bamberg,
Sept.1957-Juli 1963	Oberrealschule Bamberg,
Aug.1963-Juli 1964	Stipendiat des American Field Service an der South Williamsport-Senior-High School, Pennsylvania, USA,
Sept.1964-Juli 1966	Dientzenhofer-Gymnasium Bamberg,
Juni 1966	Abitur,
Okt.1966-Juni 1972	Studium der Mathematik und Physik an der Friedrich-Alexander-Universität Erlangen-Nürnberg,
Juni 1972	Staatsexamen,
seit Okt.1972	Verw. d. D. e. wissenschaftlichen Assistenten am Lehrstuhl für Programmiersprachen der Friedrich- Alexander-Universität Erlangen- Nürnberg,
seit 20.März 1970	verheiratet mit Monika Göttler, geb. Klein.

Offsetdruck-Fotodruck

J. Hogl, 8520 Erlangen, Hauptstr. 109 (Nähe Martin-Luther-Platz) Tel. 22990